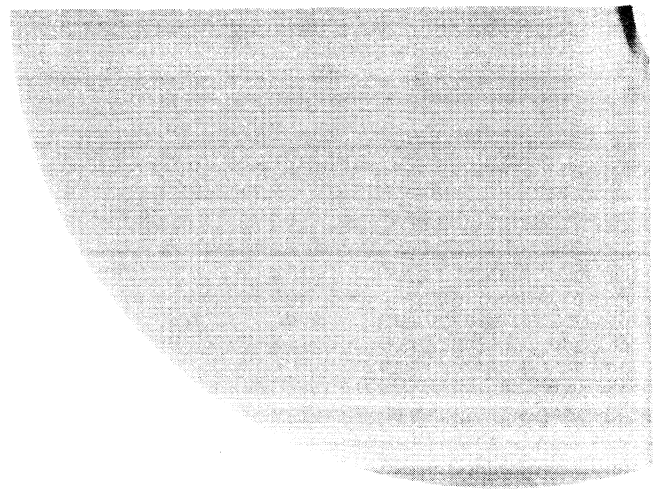
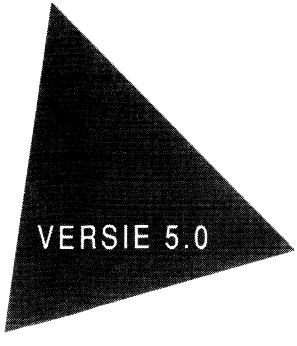


Programmeren



dBASE® voor Windows

Borland International, Inc., 100 Borland Way
P.O. Box 660001, Scotts Valley, CA 95067-0001

2E0R694

9495969798-98765432

Inhoud

Inleiding

Deze handleiding, *Programmeren*, beschrijft hoe u applicaties maakt in dBASE. *Programmeren* is vooral bestemd voor gebruikers die weinig of geen ervaring hebben met programmeren in dBASE. Voor ervaren programmeurs is dit boek een goede inleiding tot de dBASE-taal.

De handleidingen *Programmeren* en *Commando's en functies* horen bij elkaar. In deze boeken wordt de dBASE-taal beschreven en wordt informatie gegeven over de manier waarop u programma's schrijft in dBASE.

- *Commando's en functies* bevat gedetailleerde informatie over de syntaxis en functionaliteit van elk taalelement. Elk taalelement wordt afzonderlijk beschreven aan de hand van voorbeelden met code.
- *Programmeren* bevat inleidingen tot concepten, beschrijvingen van taken en voorbeelden van groepen taalelementen. Aan de hand van dit boek, en door gebruik te maken van object-georiënteerd ontwerp, kunt u dBASE-applicaties maken die actiegestuurde gebruikersinterfaces bevatten.

In Tabel I.1 worden voorbeelden gegeven van de plaatsen waar u verschillende soorten informatie over de dBASE-taal kunt vinden.

Tabel I.1 De informatie die u nodig hebt en waar u deze kunt vinden

Voorbeeld van informatie die u nodig hebt	Waar kunt u deze informatie vinden
Een voorbeeld van het gebruik van de functie TAGNO()	*Commando's en functies*
Een voorbeeld van de samenhang tussen de indexeringsfuncties	*Programmeren*
Een lijst met alle kenmerken van de klasse FORM	*Commando's en functies*
Een beschrijving van de werking van actiegestuurde programma's	*Programmeren*
De maximumlengte van de naam van een geheugenvariabele	*Commando's en functies*
Een beschrijving van de verschillende bereiken van geheugenvariabelen	*Programmeren*

De indeling van dit boek

Dit boek is opgebouwd uit de volgende delen:

- In **Deel I, "Grondbeginselen van programmeren",** worden de volgende onderdelen beschreven: grondbegrippen van programmeren in dBASE, subroutines maken, geheugenvariabelen gebruiken, werken met verschillende gegevenstypen,, programma's testen en fouten in programma's verhelpen, een preprocessor gebruiken.
- In **Deel II, "Objecten en klassen",** wordt een inleiding gegeven tot objecten, klassen en object-georiënteerd ontwerpen.
- In **Deel III, "Formulieren",** worden de volgende onderdelen beschreven: actiegestuurd programmeren, uitvoering van actie-afhandelingsroutines, onderwerpen met betrekking tot formulieren, wijzigen van code die is gegenereerd in Formulierontwerp.
- In **Deel IV, "Tabellen",** worden de volgende onderdelen beschreven: werken met tabellen, records en velden, records sorteren, programmeren voor een netwerk, transacties gebruiken, tabellen gebruiken die niet zijn gemaakt met dBASE.
- In **Deel V, "Windows-omgeving",** worden de volgende onderdelen beschreven: Windows-toepassingen, afdrukken, DLL's en de Windows API gebruiken, gegevens uitwisselen met behulp van DDE.
- Daarnaast bevat dit boek twee appendices. Deze bevatten informatie over het bijwerken van dBASE III PLUS- en dBASE IV-applicaties voor dBASE voor Windows, alsmede informatie over het werken met tekensets en taalaansturing.

Terminologie voor de muis

In Tabel I.2 worden de technieken voor het werken met de muis beschreven.

Tabel I.2 Muistechnieken

Techniek	Handeling	Gebruik
Aanwijzen	Verplaats de muisaanwijzer naar een bepaald object.	Aanwijzen gaat doorgaans vooraf aan andere handelingen.
Klikken	Druk eenmaal op de linkermuisknop en laat deze vervolgens los.	Als u de knop of het venster onder de aanwijzer wilt selecteren of als u een pictogram wilt selecteren.
Dubbelklikken	Druk tweemaal kort achter elkaar op de linkermuisknop en laat deze vervolgens los.	Als u een element wilt selecteren in een lijst of als u een pictogram wilt starten (tabel, formulier, enzovoort).
Slepen	1. Druk op de linkermuisknop en houd deze ingedrukt. 2. Verplaats de muis. 3. Laat de muisknop los.	Als u een object wilt verplaatsen of het formaat van een object wilt wijzigen. Deze techniek wordt tevens gebruikt voor *Slepen-en-neerzetten*.
Rechtsklikken	Druk eenmaal op de rechtermuisknop en laat deze vervolgens los.	Als u het kenmerkenvenster wilt weergeven voor het object onder de aanwijzer.

Opmerking Linkshandige gebruikers kunnen desgewenst de functie van de muisknoppen omwisselen in Windows. Vervang in dat geval "rechts" en "rechter" door "links" en "linker" (en omgekeerd) in Tabel I.2.

Terminologie voor het toetsenbord

Toetsnamen worden aangeduid met een speciale opmaak. Zo betekent "druk op *Alt*" dat u op de *Alt*-toets moet drukken. *Enter* verwijst naar de *Enter*-toets. Op sommige toetsenborden heeft deze toets het opschrift *Return*. In de tekst worden zoveel mogelijk de standaardnamen van de toetsen op het toetsenbord gebruikt.

In sommige gevallen moet u op twee toetsen tegelijk drukken. Dit wordt in de tekst aangeduid met een koppelteken (-) tussen de beide toetsnamen. "Druk op *Ctrl-F2*" betekent derhalve dat u *Ctrl* ingedrukt houdt terwijl u op *F2* drukt.

In de tekst wordt bovendien een speciale opmaak gebruikt om aan te geven dat u tekens moet invoeren met behulp van het toetsenbord. Bijvoorbeeld:

> Typ `SET EDITOR TO "c:\windows\notepad.exe"` in het commandovenster en druk op *Enter*.

Opmerking Als de tekenreeks aanhalingstekens bevat (in dit geval rond de padaanduiding), *moeten* deze precies zo worden getypt.

Typografische conventies

In *Programmeren* worden de verschillende elementen van de taal en de syntaxis elk met een eigen typografie aangegeven. Deze typografie wordt gebruikt om de elementen beter van elkaar te kunnen onderscheiden en het handboek beter leesbaar te maken.

Typografie	Gebruik	Voorbeelden
Cursief	Namen van variabelen en arrays, argumenten	Variabele *cNamen*, array *aKosten*, argument *<bestandsnaam>*
Kapitalen	Namen van commando's en functies, sleutelwoorden, DOS-bestanden en directory's	BROWSE, EOF(), INDEX ...TAG CUSTOMER.DBF
Beginkapitaal	Procedures, applicaties, tabelnamen, veldnamen, menu-opdrachten	Applicatie Rekeningen, veld Prijs
Kameelkapitalen	Kenmerken	OnLeftDblClk
Helvetica	Commandosyntaxis	FOUND([*alias*])
Futura smal	Namen van interface-onderdelen	Dialoogvenster **Bestand openen**
Helvetica cursief	Toetsen op het toetsenbord	Druk op *Ctrl-F10*
Typemachineletter	Codevoorbeelden	`USE Namen`

Hoofdstuk

1

Inleiding tot het programmeren in dBASE

Welkom bij dBASE, de meest produktieve taal voor het ontwikkelen van applicaties voor de Windows-omgeving. De dBASE-taal is in dBASE voor Windows aanmerkelijk verbeterd. In dit hoofdstuk worden de belangrijkste nieuwe functies van de dBASE-taal in het kort besproken en wordt de plaats in deze handleiding aangegeven waar de verschillende functies worden beschreven.

Zie Appendix A voor informatie over het starten van dBASE IV- of dBASE III PLUS-applicaties. Zie Appendix B voor informatie over het uitbreiden van deze applicaties met Windows-functies.

Object-georiënteerde uitbreidingen

Met de dBASE-taal van dBASE voor Windows kunt u applicaties object-georiënteerd ontwerpen en zijn er uitbreidingen voor het maken en gebruiken van objecten en klassen. Hoofdstuk 9 bevat een inleiding tot object-georiënteerd programmeren en in Hoofdstuk 12 wordt beschreven hoe u applicaties ontwerpt met object-georiënteerde technieken.

Objecten en klassen

Met objecten kunt u verzamelingen variabelen insluiten, waarbij sommige verzamelingen gegevens bevatten en andere verwijzende code. Met klassen kunt u groepen objecten maken die u steeds opnieuw kunt gebruiken. Met dBASE kunt u:

- Objecten maken op basis van standaardklassen of klassen die u definieert
- Eigen kenmerken toevoegen aan een object met behulp van een eenvoudige toewijzingsopdracht

- Eigen klassen definiëren.
- Klassen definiëren die zijn gebaseerd op andere klassen en die de kenmerken en methoden van die klassen overnemen
- Code schrijven die u steeds opnieuw kunt gebruiken door een klasse-hiërarchie op te zetten

In Hoofdstuk 10 wordt beschreven hoe u objecten maakt en gebruikt. In Hoofdstuk 11 wordt beschreven hoe u klassen maakt en gebruikt.

LOCAL- en STATIC-bereiken van geheugenvariabelen

dBASE IV ondersteunt twee bereiken van geheugenvariabelen: PUBLIC en PRIVATE. dBASE voor Windows voegt daar twee nieuwe bereiken aan toe: LOCAL en STATIC. Als u LOCAL- en STATIC-bereiken gebruikt, kunt u conflicten met namen van variabelen vermijden en kunt u variabelen gemakkelijker insluiten in procedures en objecten. In Hoofdstuk 5 worden de verschillende bereiken van geheugenvariabelen beschreven en vergeleken.

Codeblokken en functie-aanwijzers

In dBASE IV kunt u commando's groeperen in procedures, functies of programmabestanden. In dBASE voor Windows kunt u bovendien op twee nieuwe manieren commando's groeperen en uitvoeren:

- *Codeblok* is een nieuw gegevenstype waarmee u kleine, naamloze groepen commando's of uitdrukkingen kunt opslaan. Met codeblokken kunt u dBASE-code gebruiken als gegevens. U kunt codeblokken toewijzen aan geheugenvariabelen of objectkenmerken, en u kunt codeblokken doorgeven als parameters.
- *Functie-aanwijzer* is eveneens een nieuw gegevenstype. Hiermee kunt u verwijzingen naar procedures, functies of codeblokken opslaan. Een variabele van het type functie-aanwijzer bevat een verwijzing naar code die u kunt toewijzen aan een kenmerk, die u kunt doorgeven als een parameter of die u rechtstreeks kunt aanroepen.

Functie-aanwijzers en codeblokken zijn vooral belangrijk bij object-georiënteerd programmeren. Voor alle actiekenmerken in de standaardobjecten, zoals OnClick of OnOpen, kunt u het gegevenstype codeblok of functie-aanwijzer gebruiken. Zie Hoofdstuk 1 in *Commando's en functies* en Hoofdstuk 4 in deze handleiding voor meer informatie.

Formulieren

In dBASE voor Windows kunt u applicaties ontwerpen door de formulieren te ontwerpen waaruit de gebruikersinterface is opgebouwd.

Formuliervensters

Met een van de commando's DEFINE kunt u alle standaard weergavevensters maken voor schermen voor gegevensinvoer, dialoogvensters en meldingvensters. Als u een aantal kenmerken instelt, kunt u het formuliervenster vergroten tot maximumformaat of verkleinen tot pictogram, kunt u formulieren verplaatsen en de grootte ervan wijzigen, enzovoort. Met READMODAL() kunt u het beheer van het bureaublad overdragen aan deze formuliervensters. Met OPEN() beschikt u over een volledig actiegestuurde gebruikersinterface. In Hoofdstuk 13 worden de verschillende methoden beschreven die u kunt hanteren als u applicaties ontwerpt met formulieren.

Gebruikersinterface-objecten

dBASE bevat een volledige set grafische interface-objecten, ofwel stuurelementen, die u aan formulieren toevoegt. U kunt knoppen, keuzerondjes, aankruisvakjes, keuzelijsten, keuzelijsten met invoervak, editors, enzovoort maken met een van de bekende commando's DEFINE. U kunt ook *eigen stuurelementen* maken, waarmee u de werking van de standaard dBASE-stuurelementen kunt aanpassen. In Hoofdstuk 15 wordt beschreven hoe u eigen stuurelementen maakt en gebruikt.

Actie-afhandelingsroutines

In formuliervensters en alle daarin opgenomen objecten worden *acties* (handelingen die door de gebruiker of het systeem worden uitgevoerd) automatisch herkend en gevolgd door een reactie. U kunt bepalen hoe een formulier op acties reageert door *actie-afhandelingsroutines* te schrijven. Dit zijn procedures die worden uitgevoerd wanneer een actie zich voordoet. Zie Hoofdstuk 2 voor een inleiding tot de werking van actiegestuurde programma's. Zie Hoofdstuk 14 voor meer informatie over het schrijven van actie-afhandelingsroutines.

Tweeweghulpmiddelen

Formulierontwerp en Menu-ontwerp zijn veelzijdige hulpmiddelen waarmee u code genereert en waarmee u op visuele wijze formulieren en menu's ontwerpt. Deze hulpmiddelen maken dBASE-programmabestanden die u kunt wijzigen met een teksteditor en die u daarna opnieuw in de hulpmiddelen kunt laden. Hierdoor kunt u programma's ontwerpen zoals u dat wilt, waarbij u zo nodig objecten kunt indelen, de code kunt verfijnen en naar Formulier- of Menu-ontwerp kunt teruggaan. In Hoofdstuk 16 wordt beschreven hoe u de gegenereerde code gebruikt.

Sessies

Met een nieuw commando, CREATE SESSION, kunt u tabellen in formulieren insluiten door een nieuwe set werkgebieden (met afzonderlijke recordaanwijzers) te maken die exclusief worden toegewezen aan het formulier, op dezelfde wijze als wanneer u nog een dBASE-sessie had gestart. Een sessie in een actiegestuurde omgeving, waarin u kunt schakelen tussen taken en naar willekeur tabellen kunt openen en sluiten, kan worden

vergeleken met een afzonderlijke aanmelding in een netwerk. Zie Hoofdstuk 21 voor meer informatie over het programmeren voor een netwerkomgeving.

Windows-omgeving

De Windows-omgeving bevat vele functies (Dynamic Data Exchange (DDE); Object Linking and Embedding (OLE)) en gemeenschappelijk te gebruiken resources (printeraansturingsprogramma's en fonts) waarmee u uw applicaties kunt uitbreiden.

Printeraansturingsprogramma's voor Windows

DOS-toepassingen maken doorgaans gebruik van eigen aansturingsprogramma's voor printers. In de Windows-omgeving beschikt u over ingebouwde printeraansturingsprogramma's die door alle Windows-toepassingen kunnen worden gedeeld. Hierdoor is het niet nodig dat u voor elke toepassing die u installeert, printeraansturingsprogramma's moet configureren.

dBASE voor Windows ondersteunt de printeraansturingsprogramma's voor Windows en maakt gebruik van Windows-voorzieningen voor het afdrukken, zoals Afdrukbeheer. Tevens biedt dBASE voor Windows ondersteuning voor de meeste afdrukopdrachten en systeemgeheugenvariabelen uit dBASE IV.

Met een nieuwe functie, CHOOSEPRINTER(), kunt u een dialoogvenster voor de printerinstelling weergeven dat door de meeste Windows-toepassingen wordt gebruikt. In dit dialoogvenster kunt u een andere printer selecteren, de paginarichting (liggend of staand) instellen en de wijze van papierinvoer opgeven. De instellingen van systeemgeheugenvariabelen, zoals _pdriver en _porientation, worden zo nodig automatisch bijgewerkt door de instellingen die u in dit dialoogvenster opgeeft. Zie Hoofdstuk 24 voor meer informatie.

Windows-fonts

Evenals bij printeraansturingsprogramma's het geval is, hebben DOS-toepassingen vaak speciale fonts nodig voor het afdrukken. De Windows-omgeving ondersteunt veelgebruikte fonts die door alle Windows-toepassingen kunnen worden gedeeld.

dBASE ondersteunt Windows-fonts. U kunt tekst weergeven of afdrukken met behulp van de geïnstalleerde fonts en u kunt het uiterlijk van schermtekst wijzigen met behulp van fontkenmerken, zoals FontBold, FontItalic of FontUnderline. In Hoofdstuk 8 van *Commando's en functies* worden de fontkenmerken beschreven.

DLL's en aanroepen van de Windows API

Dynamic Link Libraries (DLL's) zijn bestanden die gecompileerde programmacode bevatten, die tijdens runtime door Windows-toepassingen wordt geladen en uitgevoerd. De programmacode kan subroutines en resources bevatten, zoals bitmaps en pictogrammen. Met behulp van DLL's kunt u niet-dBASE-functies aanroepen vanuit dBASE-code.

De Windows API (Application Programming Interface) is een bibliotheek die deel uitmaakt van de Windows-omgeving en die honderden bruikbare C-functies bevat, die zijn opgeslagen in DLL's. Doordat DLL wordt ondersteund, kunnen programmeurs in dBASE toegang verkrijgen tot de Windows API.

DLL-functies kunt u gemakkelijk interactief of vanuit een programma benaderen. U maakt eenvoudig een prototype van een functie met het commando EXTERN. Vervolgens roept u de functie aan alsof deze tot de dBASE-taal behoort. Zie Hoofdstuk 25 voor meer informatie.

Multimedia: grafische afbeeldingen en geluid

U kunt veelgebruikte indelingen voor Windows-multimediabestanden voor geluid en grafische afbeeldingen, zoals .WAV, .BMP en .PCX, maken, bewerken, afspelen en weergeven. Gebruik RESTORE IMAGE als u een grafische afbeelding wilt weergeven en gebruik PLAY SOUND als u geluiden uit een .WAV-bestand wilt afspelen. U kunt geluidsbestanden en grafische bestanden ook opslaan in het binaire veld van een tabel of u kunt deze bestanden weergeven in formulieren.

Zie *Commando's en functies* voor meer informatie over de commando's RESTORE IMAGE en PLAY SOUND. Zie Hoofdstuk 18 in deze handleiding voor meer informatie over het opslaan van binaire gegevens in tabellen.

DDE

Dynamic Data Exchange (DDE) is een Windows-voorziening waarmee twee toepassingen gegevens en instructies kunnen delen en uitwisselen. dBASE voor Windows ondersteunt DDE als client- en als server-toepassing. Zie Hoofdstuk 26 voor meer informatie over DDE.

Met een andere voorziening voor gegevensuitwisseling, Object Linking and Embedding (OLE), kunt u externe toepassingen rechtstreeks gebruiken vanuit een dBASE-tabel of -formulier. Zie het *Handboek* voor meer informatie over OLE.

Tabellen

Tabellen zijn de belangrijkste middelen voor gegevensopslag in dBASE voor Windows. In vorige versies van dBASE worden tabellen *databasebestanden* genoemd. In dBASE voor Windows wordt één gegevensbestand een tabel genoemd en wordt een verzameling tabellen en bijbehorende bestanden een database genoemd. De bestanden zelf blijven ongewijzigd ten opzichte van dBASE IV, alleen de terminologie is gewijzigd.

dBASE voor Windows bevat verscheidene nieuwe voorzieningen voor het werken met tabellen in programma's.

IDAPI en de Borland Database Engine (BDE)

Behalve toegang tot tabellen met de DBF-indeling van dBASE zelf, biedt dBASE ook volledige toegang tot tabellen met andere indelingen. U kunt Paradox- en SQL-tabellen

openen, maken, wijzigen en een query op deze tabellen uitvoeren met dezelfde commando's die u gebruikt voor standaard dBASE-tabellen. U kunt zelfs relaties leggen tussen verschillende bestandstypen. Met behulp van IDAPI en Borland Database Engine wordt de toegang tot verschillende typen tabellen zo volledig mogelijk gemaakt, zonder dat een aparte laag of taal wordt toegevoegd voor het bewerken van de gegevens. Zie Hoofdstuk 23 voor meer informatie over het werken met Paradox- en SQL-tabellen.

Ondersteuning van SQL-taal

Als u een SQL-tabel opent, wordt de server benaderd en wordt vervolgens een SQL-instructie uitgevoerd waarmee records worden opgehaald. De opgehaalde gegevens worden opgeslagen in een lokale buffer, zodat u de gegevens kunt weergeven en bewerken. Als u de gegevens niet hoeft te bewerken, kunt u SQL-instructies rechtstreeks naar de server sturen met de functie SQLEXEC(). Als u een SQL-instructie rechtstreeks naar de server stuurt, kunt u elke bewerking uitvoeren die wordt ondersteund door de database-server die u benadert. Zie Hoofdstuk 23 voor meer informatie.

Referentiële integriteit

Met nieuwe opties van het commando SET RELATION kunt u strikte referentiële integriteit afdwingen tussen gerelateerde .DBF-tabellen. Met de optie CONSTRAIN beperkt u de toegang van de subtabel tot de records die overeenkomen met de hoofdtabel. Hierdoor hoeft u geen extra filter te gebruiken voor de subtabel. Met de optie INTEGRITY CASCADE kunt u automatisch subrecords verwijderen als het overeenkomende hoofdrecord wordt verwijderd. Met de optie INTEGRITY RESTRICTED kunt u voorkomen dat subrecords worden verwijderd als het overeenkomende hoofdrecord niet is verwijderd. Zie Hoofdstuk 19 voor meer informatie.

Actiegestuurde transactieverwerking

De traditionele transactieverwerking van dBASE IV werkt goed in procedurele applicaties, maar niet in *actiegestuurde* omgevingen als Windows. In dBASE voor Windows kunt u de transactiecommando's van dBASE IV niet gebruiken. In plaats daarvan maakt dBASE gebruik van actiegestuurde transactieverwerking, waarbij de uitvoering van bewerkingen voor het terugdraaien en doorvoeren van transacties niet afhangt van het codeblok waar de transacties zijn begonnen. Met drie nieuwe functies, BEGINTRANS(), COMMIT() en ROLLBACK(), kunt u transactiepunten opgeven. Zie Hoofdstuk 22 voor meer informatie.

Nieuwe veldtypen: Binair en OLE

dBASE bevat nieuwe veldtypen voor het opslaan van binaire gegevens en OLE-gegevens in .DBF-tabellen. Deze nieuwe typen vormen een aanvulling op de reeds bestaande memovelden. Niet alleen kunt u deze velden toevoegen aan tabellen, maar bovendien kunt u met een aantal nieuwe commando's, zoals COPY BINARY,

REPLACE BINARY, REPLACE OLE, RESTORE IMAGE en PLAY SOUND, gegevens opslaan in en ophalen uit deze velden. In Hoofdstuk 18 worden hiervan voorbeelden gegeven.

Bladwijzers

Paradox- en SQL-tabellen hebben geen vaste recordnummers zoals dBASE-tabellen. Met *bladwijzers* kunt u Paradox-, SQL- en dBASE-tabellen de functionaliteit van dBASE-recordnummers geven. De functie BOOKMARK() stelt een aanwijzer naar een record in die u op dezelfde wijze kunt gebruiken als een recordnummer. Bladwijzers hebben als voordeel dat de waarden ervan altijd zijn gebaseerd op de huidige volgorde van de records in een tabel. Als u derhalve de waarden van BOOKMARK() vergelijkt met behulp van >, = of < krijgt u de relatieve positie van twee records in de huidige indexvolgorde. Zie Hoofdstuk 23 voor meer informatie.

Memovelden in uitdrukkingen

Memovelden die tekst bevatten, kunt u gebruiken op alle plaatsen waar een tekenuitdrukking is toegestaan. De enige uitzondering vormt het commando INDEX (u kunt geen index maken op basis van een memoveld). Hierdoor kunt u memovelden gemakkelijk bewerken en indelen met de functies voor tekenreeksafhandeling, zoals UPPER(), TRANSFORM() of STUFF(). In Hoofdstuk 18 worden hiervan voorbeelden gegeven.

Programmeren (algemeen)

Hierna worden enkele van de nieuwe programmeerfuncties besproken die zijn toegevoegd aan dBASE.

Preprocessor

dBASE voor Windows bevat een ingebouwde preprocessor die lijkt op de preprocessoren die worden gebruikt in compileerprogramma's voor de C-taal. Als u een programma compileert, doorzoekt de preprocessor de code op *preprocessor-instructies* en evalueert deze. Daarbij wordt een tijdelijk tussenbestand gemaakt dat door het compileerprogramma wordt gecompileerd. Als u preprocessor-instructies gebruikt in uw code, kunt u programmabestanden invoegen, constanten definiëren, uitdrukkingen uitbreiden en voorwaardelijke compilaties uitvoeren. Met de preprocessor hebt u meer controle over het compileerproces. Zie Hoofdstuk 7 voor meer informatie.

Debugger

De debugger bevat een volledige set hulpmiddelen voor het testen van programma's op fouten. Controlepunten, afbreekpunten, broncode en de aanroepende stapel kunnen afzonderlijk of tegelijkertijd worden bekeken. Met een krachtig controle-instrument

kunt u variabelen, velden en objecten bekijken en de waarden ervan wijzigen tijdens de uitvoering. Zie Hoofdstuk 8 voor meer informatie over het gebruik van de debugger.

Krachtiger array's

In dBASE voor Windows beschikken array's over krachtige nieuwe mogelijkheden. U kunt:

- Array's definiëren met een willekeurig aantal dimensies
- Array-elementen toevoegen, verwijderen, sorteren en doorzoeken met nieuwe functies als AINS(), ADEL(), ASORT(), ASCAN(), enzovoort
- Met array's werken als met objecten door de bijbehorende kenmerken en methoden te benaderen
- *Minimale array's* maken met een variabel aantal niet-aaneengesloten elementen

Zie Hoofdstuk 5 voor meer informatie over de nieuwe functies voor array's. In Hoofdstuk 10 wordt beschreven hoe u array's als objecten gebruikt en hoe u minimale array's maakt.

Uitbreidingen voor doorgeven van parameters

Het doorgeven van parameters aan procedures en functies is aanzienlijk uitgebreid. U kunt:

- Maximaal 255 parameters doorgeven aan een procedure
- Complete array's en objecten doorgeven
- Meer of minder parameters doorgeven dan de parameters die zijn gedefinieerd voor de procedure of functie
- Parameters doorgeven als waarde of als verwijzing

Zie Hoofdstuk 4 voor meer informatie.

Dekkingsanalyse

Dekkingsanalyse is een krachtige functie die precies aangeeft welke secties met code worden uitgevoerd wanneer u een programma start. Deze gegevens zijn van essentieel belang wanneer u testreeksen wilt verbeteren en ervoor wilt zorgen dat alle delen van een applicatie worden getest. Dekkingsanalyse is een van de weinige analysehulpmiddelen die niet ingrijpen in code voor talen voor applicatie-ontwikkeling. Zie Hoofdstuk 3 voor meer informatie.

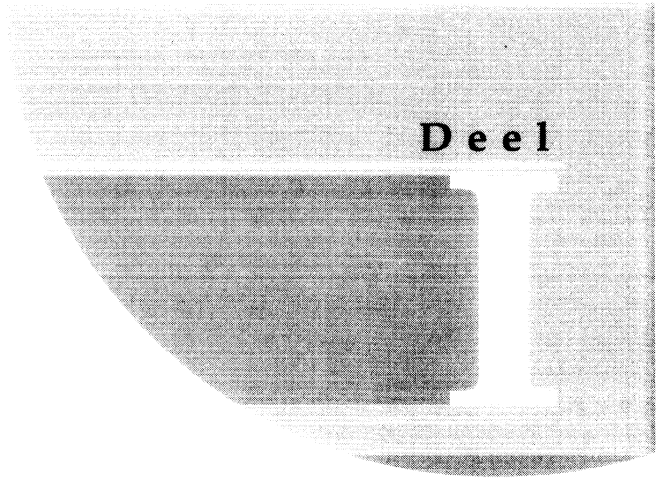
Genereren van willekeurige gegevens voor testdoeleinden

Met het nieuwe commando GENERATE kunt u een tabel op eenvoudige wijze opvullen met willekeurige gegevens voor testdoeleinden. U hoeft alleen de tabel te openen en het commando GENERATE te geven samen met een numeriek argument waarmee u

aangeeft hoeveel records u wilt maken. Als u bijvoorbeeld `GENERATE 100` opgeeft, worden alle velden (met uitzondering van memovelden, binaire velden en OLE-velden) opgevuld met willekeurige gegevens van het juiste type voor de eerste 100 records. Zie Hoofdstuk 3 voor meer informatie.

Verhoogde capaciteit

De capaciteit die u bij diverse handelingen ter beschikking hebt, zoals het aantal velden in een tabel, het aantal actieve werkgebieden en de lengte van namen van geheugenvariabelen, is verhoogd ten opzichte van dBASE IV. Zie Appendix B in *Commando's en functies* voor een overzicht.



Grondbeginselen van programmeren

In dit deel worden actiegestuurde programma's besproken, alsmede algemene programmeerapecten van dBASE.

Is dit uw eerste kennismaking met programmeren in dBASE, dan begint u met de volgende hoofdstukken:

1 Hoofdstuk 2, als algemene inleiding.

2 Hoofdstuk 3, met name de secties over het maken en compileren van programma's.

3 Hoofdstuk 4, 5, en 6, over de grondbeginselen van het schrijven van dBASE-code.

Dit deel bevat de volgende hoofdstukken:

- Hoofdstuk 2, "Aan de slag met programmeren in dBASE"
- Hoofdstuk 3, "Programma's maken, compileren en testen"
- Hoofdstuk 4, "Werken met procedures en codeblokken"
- Hoofdstuk 5, "Werken met geheugenvariabelen"
- Hoofdstuk 6, "Werken met gegevenstypen"
- Hoofdstuk 7, "Werken met preprocessor-instructies"
- Hoofdstuk 8, "Fouten verhelpen in programma's"

Hoofdstuk

2

Aan de slag met programmeren in dBASE

Dit hoofdstuk bevat een inleiding tot actiegestuurde programma's, een beschrijving van de werking van deze programma's en een overzicht van het proces van het maken van actiegestuurde programma's met behulp van dBASE voor Windows.

Zie Appendix A voor meer informatie over het starten van dBASE III PLUS- of dBASE IV-applicaties.

Programma's schrijven vs. visueel programmeren

Met een tekst-editor kunt u complete programma's schrijven door de commando's regel voor regel te typen. Op deze manier schreven de meeste dBASE-programmeurs hun programma's. In dBASE voor Windows gebruikt u ontwerphulpmiddelen waarmee u programmacode automatisch kunt genereren. Het meest problematische onderdeel van het traditionele programmeren in dBASE, raden hoe velden en menu's verschijnen nadat u de positie ervan hebt opgegeven met coördinaten, is vervallen. In dBASE voor Windows plaatst u objecten precies op de gewenste plaats in een formulier en de coördinaten worden automatisch berekend. Dat is nu visueel programmeren.

Visueel programmeren omvat echter meer dan het bepalen van de indeling van formulieren. De objecten die u in formulieren plaatst, kunnen reageren op handelingen van de gebruiker. Een muisklik wordt automatisch herkend door een knop. Een formulier "weet" wanneer het door de gebruiker wordt verplaatst, of vergroot of verkleind. U hoeft niet uit te zoeken wat de gebruiker doet en hoe dat in zijn werk gaat. Daar zorgt dBASE voor. U hoeft alleen maar in te stellen hoe de objecten op deze acties moeten reageren. Hiertoe wijst u procedures toe die worden uitgevoerd wanneer de acties zich voordoen.

Wilt u gegevens met een verschillende letterstijl of lettergrootte weergeven? U hoeft dan alleen maar het object te selecteren en de bijbehorende fontkenmerken in te stellen. Wilt

u een grafische afbeelding weergeven in een formulier? Sleep dan een object van het type Afbeelding van het palet in het formulier en koppel dit object aan een bitmapbestand of een memoveld.

In Hoofdstuk 14 wordt een beknopt overzicht gegeven van de manier waarop u actieafhandelingsroutines schrijft en deze koppelt aan acties met behulp van Formulierontwerp. Zie Hoofdstuk 8 van het *Handboek* voor een algemene inleiding in Formulierontwerp en de bijbehorende hulpmiddelen.

Visueel programmeren vereenvoudigt het maken van applicaties aanzienlijk. Bovendien zijn de applicaties eenvoudig in het gebruik omdat ze actiegestuurd zijn.

Het waarom van actiegestuurde programma's

De drie belangrijkste typen gebruikersinterfaces zijn:

- *Opdrachtregel.* Hierbij typt de gebruiker opdrachten achter een prompt. Het besturingssysteem MS-DOS, de commandostip van dBASE (in vorige versies) en het commandovenster (in dBASE voor Windows) zijn voorbeelden van opdrachtregelinterfaces.
- *Menu-gestuurd.* Hierbij selecteert de gebruiker keuzemogelijkheden in een hiërarchie van menu-onderdelen. De meeste applicaties die zijn geschreven met vorige versies van dBASE, hebben menu-gestuurde interfaces.
- *Actiegestuurd.* Hierbij reageert de gebruiker op zichtbare objecten, zoals formulieren die knoppen en keuzelijsten bevatten, in willekeurige volgorde. De gebruikersinterface van dBASE voor Windows is actiegestuurd. Met de dBASE-taal kunt u bovendien zelf actiegestuurde applicaties ontwerpen.

In dBASE kunt u met behulp van traditionele programmeertechnieken, zoals de menu- en vensteropdrachten van dBASE IV, actiegestuurde interfaces ontwerpen. In deze applicaties wordt de opeenvolging van acties bepaald door het programma. Als de gebruiker bijvoorbeeld **Orders invoeren** kiest in een menu, verschijnt achtereenvolgens een aantal schermen waarin de gebruiker gegevens moet typen, zoals de naam van de klant, de orderdatum, het ordernummer, de artikelen en de verzendkosten.

Deze menu-gestuurde technieken zijn niet goed bruikbaar voor het programmeren in actiegestuurde omgevingen als Windows. In een actiegestuurde applicatie bepaalt de *gebruiker* de opeenvolging van programma-onderdelen. Een gebruiker kan bijvoorbeeld op elk gewenst moment op een knop klikken, een venster activeren of een keuze maken uit een menu. Het programma dient op deze acties te reageren in de volgorde waarin deze plaatsvinden.

Voor elke actiegestuurde applicatie met een deugdelijk ontwerp geldt het volgende:

- *De gebruiker kan zich concentreren op de taak en hoeft zich niet te concentreren op de applicatie.* De gebruiker hoeft zich geen complexe hiërarchie van menukeuzen eigen te maken. Wanneer de gebruiker bijvoorbeeld een order wil invoeren, verschijnt een orderformulier dat veel weg heeft van een standaard papieren formulier.
- *De gebruiker hoeft niet opnieuw te leren hoe taken moeten worden uitgevoerd.* U maakt de interface van een applicatie met behulp van bekende objecten als knoppen en

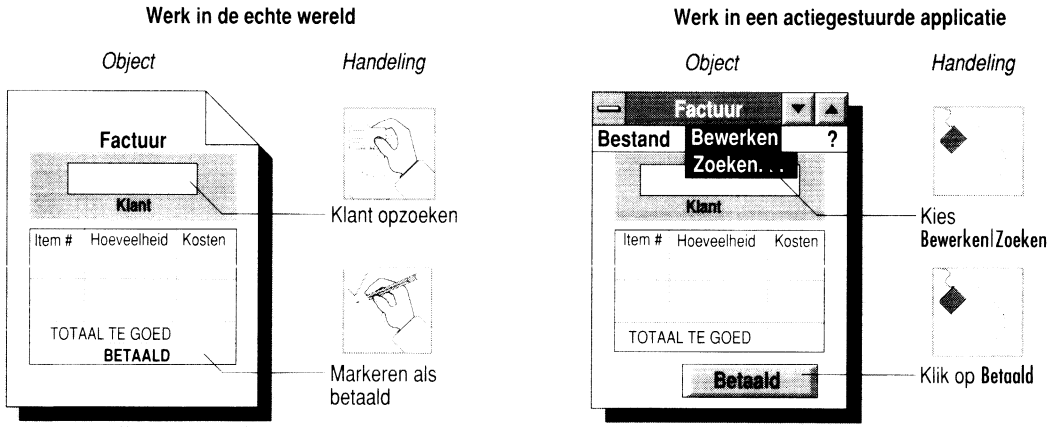
keuzelijsten. Hierdoor verlopen veelvoorkomende bewerkingen, zoals een bestand openen, navigeren in een formulier en de applicatie afsluiten, grotendeels hetzelfde in verschillende applicaties.

Het belangrijkste is echter dat actiegestuurde applicaties een weergave vormen van de manier waarop mensen in het echt te werk gaan. Als verkoopmedewerkers een order opnemen, pakken zij een formulier en vullen dit in. Wanneer zij een cheque ontvangen voor een order, pakken zij de desbetreffende factuur en markeren deze als betaald. Deze opeenvolging van werkzaamheden volgt een patroon van *object-handeling*. Dat wil zeggen, een medewerker selecteert een object (een orderformulier, een factuur) en voert er een handeling mee uit (vult het formulier in, post de cheque).

Op dezelfde wijze selecteert een gebruiker in een actiegestuurde applicatie objecten (formulieren, invoervakken, knoppen) en voert er handelingen mee uit.

In Afbeelding 2.1 wordt het object-handeling-patroon van werkzaamheden in de echte wereld en in actiegestuurde applicaties toegelicht.

Afbeelding 2.1 Het werkpatroon object-handeling



De werking van actiegestuurde programma's

In een dBASE voor Windows-applicatie is het formulier de belangrijkste component van de gebruikersinterface. Formulieren bevatten *stuurelementen*, zoals invoervakken en knoppen, voor de interactie met de gebruiker. *Acties, zoals klikken met de muis of drukken op een toets, worden door stuurelementen herkend.* In reactie op acties worden *actie-afhandelingsroutines*, ofwel door u geschreven programmacodes, uitgevoerd.

U verbindt code met kenmerken van actie-afhandelingsroutines voor stuurelementen, zoals OnClick of OnLeftMouseDown (de meeste beginnen met On), die corresponderen met bepaalde acties. Als een gebruiker bijvoorbeeld op een knop klikt, wordt de actie-afhandelingsroutine OnClick uitgevoerd.

Het opgeven van actie-afhandelingsroutines voor formulieren komt overeen met het gebruik van de commando's ON in dBASE IV, zoals ON KEY LABEL of ON ERROR.

Evenals een actie-afhandelingsroutine wijst het commando ON programmacode toe die wordt uitgevoerd wanneer een actie, zoals het drukken op een toets of een runtime-fout, plaatsvindt. De acties die worden afgehandeld door de commando's ON zijn echter beperkt en zijn niet gekoppeld aan objecten van een gebruikersinterface.

Bij de meeste actiegestuurde applicaties worden de volgende stappen uitgevoerd:

1 De applicatie geeft automatisch een beginformulier weer.

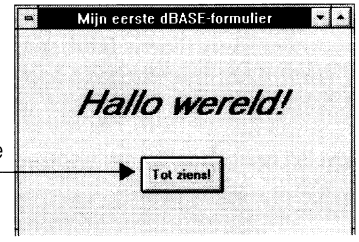
2 Een actie, zoals een muisklik of een toetsaanslag, vindt plaats met betrekking tot het formulier of een stuurelement in het formulier.

3 Een actie-afhandelingsroutine wordt uitgevoerd die is gekoppeld aan de actie uit stap 2.

4 De applicatie wacht op de volgende actie.

In Afbeelding 2.2 ziet u het voorbeeldformulier "Hallo wereld!". Als het formulier is geopend, wacht dBASE op een actie. De gebruiker kan het formulier verplaatsen, het formaat ervan wijzigen, het verkleinen tot pictogram of het vergroten tot maximumvenster. Als de gebruiker op de knop klikt, wordt de actie OnClick uitgevoerd.

Afbeelding 2.2 Voorbeeld van actie-afhandelingsroutines voor het formulier "Hallo wereld!"



In de volgende code, een .WFM-bestand dat wordt gegenereerd door Formulierontwerp, wordt het formulier uit Afbeelding 2.2 gemaakt. Deze code heeft dezelfde structuur als de code van alle formulieren die in Formulierontwerp worden gegenereerd. Voorlopig hoeft u niet elke regel te begrijpen. Het is voldoende wanneer u de algemene structuur bekijkt, zodat u een indruk krijgt van de manier waarop formulieren worden gemaakt, kenmerken worden ingesteld en actie-afhandelingsroutines worden toegewezen aan acties.

```
** END HEADER -- deze regel niet verwijderen*
* gemaakt op 29-09-94                    && Automatisch datumstempel
*
LOCAL f                                  && LOCAL-variabele, verwijzing naar
                                         && formulierobject
f = NEW HALLOFORM()                      && Formulier maken
f.Open()                                 && Formulier openen

CLASS HALLOFORM OF FORM                  && HALLOFORM definiëren als klasse van FORM
   this.EscExit = .T.                    && Esc inschakelen om formulier te sluiten
   this.Text = "Mijn eerste dBASE-formulier"
                                         && Titel formuliervenster
```

```
this.Width =          63.60              && Breedte formulliervenster
this.Top =          14.39                && Positie bovenzijde formulliervenster
this.Left =          54.86               && Positie linkerzijde formulliervenster
this.Height =         15.15              && Hoogte formulliervenster
this.Minimize = .T.                      && Verkleinen tot pictogram inschakelen
this.Maximize = .T.                      && Vergroten tot maximumvenster inschakelen

DEFINE TEXT TEKST1 OF THIS;              && Tekstobject maken
    PROPERTY;
       ColorNormal "N/W",;
       Text "Hallo wereld!",;            && Tekst is "Hallo wereld"
       Width          36.57,;           && Afmetingen tekstvak volgen
       Top           3.03,;
       Left           12.72,;
       Height          3.03,;
       FontSize         23.00,;         && Grootte standaardfont
       FontItalic .T.                    && Cursief

DEFINE PUSHBUTTON KNOP1 OF THIS;         && Knop-object maken
    PROPERTY;
       OnClick CLASS::KNOP1_ONCLICK,;    && Actie-afhandelingsroutine toewijzen aan knop
       StatusMessage "Klik op de knop als u het formulier wilt sluiten.",;
                                         &&Melding voor gebruiker
       Text "Tot ziens",;
       Width           20.67,;           && Afmetingen knop volgen
       Top           9.09,;
       Left         20.67,;
       Height          2.02,;
       Group .T.

PROCEDURE KNOP1_OnClick                  && Actie-afhandelingsroutine voor actie OnClick
   DO WHILE form.Height > 0 .and. form.Width > 0
      form.Tekst1.Text = "Tot ziens!"
      form.Height = form.Height - 1
      form.Width = form.Width - 1
      form.Top = form.Top + .5
      form.Left = form.Left + .5
      DO MaakEenGeluid
   ENDDO
   form.Close()
RETURN
ENDCLASS                                 && Klassedefinitie beëindigen
```

Actiegestuurde programma's ontwerpen

Alles wat u werkelijk nodig hebt voor het ontwerpen van actiegestuurde programma's is Formulierontwerp. Met Formulierontwerp en de bijbehorende hulpmiddelen kunt u formulieren voor gegevensinvoer, dialoogvensters en menu's maken. Kortom, alle zichtbare componenten van een applicatie. Met de procedure-editor (een hulpmiddel van Formulierontwerp waarmee u programmacode invoert en bewerkt) schrijft u

vervolgens procedures die worden uitgevoerd wanneer acties plaatsvinden. Op deze wijze koppelt u de componenten aan elkaar.

Bij meer complexe projecten kunt u echter niet zonder een planning en een goed ontwerp. Hiervoor kunt u gebruik maken van object-georiënteerde technieken. Met deze technieken kunt u gerelateerde informatie groeperen in objecten die u zelf hebt gedefinieerd, kunt u "klassen" van gerelateerde objecten samenstellen en kunt u nieuwe objecten maken door eenvoudige wijzigingen aan te brengen in bestaande objecten.

In Hoofdstuk 12 wordt een inleiding gegeven tot object-georiënteerde technieken voor het ontwerpen van applicaties.

Hoofdstuk

3

Programma's maken, compileren en testen

dBASE voor Windows ondersteunt zowel procedureel programmeren met een tekst-editor als interactief visueel programmeren met Formulierontwerp en Query-ontwerp. U kunt de beide programmeermethoden in hetzelfde programma combineren en u kunt code die is gegenereerd in Formulierontwerp, integreren in de traditionele programma-componenten van dBASE. Op deze wijze kunt u applicaties maken met een gebruikersinterface in de stijl van Windows.

In dit hoofdstuk worden de volgende onderwerpen beschreven: programmabestanden maken, stijlconventies voor het programmeren, compileren met automatische vergelijking van bestandsdatums, compileren met dekkingsanalyse en programma's testen.

Een programmabestand maken

Een dBASE-programma is een tekstbestand dat dBASE-opdrachten bevat. De opdrachten in een programmabestand worden *programmacode*, broncode of gewoon *code* genoemd.

Met de tekst-editor van dBASE kunt u programma's schrijven. Met het commando CREATE/MODIFY COMMAND opent u een editor-venster waarin u nieuwe code kunt invoeren of bestaande code kunt wijzigen.

Als u een naamloos editor-venster wilt openen in de tekst-editor van dBASE, typt u het commando CREATE/MODIFY COMMAND of het commando CREATE/MODIFY FILE in het commandovenster. Voorbeeld:

```
MODIFY FILE CONTACT.PRG
```

De code voor het programma Contact wordt als tekstbestand in een venster geopend.

Stijlconventies gebruiken

U bepaalt de opmaak van de tekstbestanden met programmacode aan de hand van regels voor inspringing, naamgeving en gebruik van hoofdletters, zodat de code gemakkelijk te lezen is. Vooral in grote programma's gebruikt u bij minder duidelijke gedeelten commentaarregels om uit te leggen wat er gebeurt. Hierdoor zijn programma's gemakkelijk te begrijpen en bij te werken, vooral wanneer dit laatste door verschillende mensen wordt gedaan.

Waarschijnlijk ontwikkelt u na verloop van tijd uw eigen stijl voor het schrijven van programmacode. Hierna volgen enkele conventies waaraan u zich dient te houden:

- *Voeg commentaar toe aan uw code.* Naast dBASE-commando's kan een programmabestand andere tekst bevatten waarmee u de werking van uw code beschrijft. dBASE herkent drie verschillende markeringen voor commentaar:
  - Twee en-tekens (&&)
  - Asterisk (*), maar alleen als eerste teken van een regel
  - Het commando NOTE, in hoofdletters of kleine letters, en alleen aan het begin van een regel

```
USE LANDEN EXCLUSIVE          && Open de tabel Landen
INDEX ON Naam TAG Naam
* Maak een indexlabel
BROWSE
note Blader door de tabel
```

- *Neem een inleiding tot het programma als commentaar op.* U kunt een of meer alinea's met commentaar opnemen aan het begin van een programmabestand als een korte beschrijving van het programma. Nuttige informatie is bijvoorbeeld:
  - Revisiedatum, naam van de programmeur en doel van het programma
  - Het besturingssysteem waaronder het programma draait: DOS, Windows of een ander systeem
  - De syntaxis voor de opdrachtregel, en eventuele schakelopties
- *Laat code inspringen.* Als u regels met code laat inspringen, geeft u de lezer een visuele aanwijzing voor de hiërarchie van programma-opdrachten en kunnen geneste lussen goed worden onderscheiden. Als u code laat inspringen, worden programma-componenten zonder overeenkomende begin- en eindopdracht duidelijk weergegeven.

```
IF Naam = "Sandra"
   ? "Hallo Sandra!"
ELSE
   IF Naam = "Jan"
      ? "Hallo Jan!"
   ENDIF
ENDIF
```

- *Verdeel lange opdrachten in kleinere stukken.* Met de tekst-editor van dBASE kunt u 1024 bytes op een coderegel gebruiken. Dit is echter geen handige regellengte voor schermweergave of afdrukken op de printer. U kunt lange code-opdrachten in

stukken verdelen met de puntkomma (;), het vervolgteken in dBASE voor Windows. Als u een puntkomma aan het einde van een programmaregel plaatst en vervolgens op *Enter* drukt of een regelterugloopteken gebruikt, gaat dBASE door naar de volgende regel voordat de programma-opdracht wordt geïnterpreteerd.

- *Wees consistent in het gebruik van namen en hoofdletters voor taalelementen.* In dBASE-taal wordt geen onderscheid gemaakt tussen hoofdletters en kleine letters, dus u kunt hoofdletters gebruiken als u de leesbaarheid van geheugenvariabelen en veldnamen wilt vergroten. Dit is echter niet noodzakelijk. Als u altijd hebt geprogrammeerd in C en u gewend bent aan het maken van onderscheid tusen hoofdletters en kleine letters, wilt u waarschijnlijk geen hoofdletters gebruiken.

  Begin geheugenvariabelen met dezelfde tekens. In dat geval kunt u namelijk met jokertekens zoeken naar namen van geheugenvariabelen, die u dan gemakkelijk en op volgorde terugvindt.

- *Gebruik een notatie die het bereik en het type van geheugenvariabelen aanduidt.* Er zijn vier verschillende bereiken voor geheugenvariabelen in dBASE: LOCAL, STATIC, PUBLIC en PRIVATE. Deze kunnen worden gecombineerd met de gegevenstypen Teken, Numeriek, Datum en Logisch als u uit twee letters bestaande voorvoegsels wilt maken voor inzichtelijke namen van geheugenvariabelen. Bijvoorbeeld:

```
ll_betaald          && LOCAL logische variabele
sn_BalansOud        && STATIC numerieke variabele
lc_TijdBestand      && LOCAL teken-variabele
```

Opmerking In dBASE beginnen vooraf gedefinieerde namen van variabelen met een onderstreepteken. Deze variabelen worden systeemgeheugenvariabelen genoemd. Als u wilt voorkomen dat u de naam van een systeemgeheugenvariabele voor een eigen variabele gebruikt, moet u de namen van uw variabelen niet laten beginnen met een onderstreepteken.

Programma's compileren

Nadat u een programma hebt geschreven, moet u het tekstbestand compileren zodat een objectbestand met code wordt gemaakt. In dit hoofdstuk verwijst compileren en starten alleen naar programma's die werken in de geïnterpreteerde dBASE-omgeving. Deze gecompileerde objectbestanden bevatten pseudo-code (ook wel *pcode* of tokens genoemd). Voor deze programma's is het noodzakelijk dat dBASE actief is. Deze programma's verschillen van programma's die zijn gecompileerd op het niveau van het besturingssysteem en die kunnen worden gestart als standalone-toepassingen op computers waarop dBASE niet is geïnstalleerd.

Afhankelijk van het type code dat u schrijft of genereert met de ontwerphulpmiddelen, gebruikt dBASE verschillende standaard-bestandsextensies. Omdat deze bestandsextensies van belang zijn voor het compileerprogramma en betrekking hebben op relaties tussen broncode en gecompileerde code, moet u de extensies niet wijzigen.

Gebruik voor namen van bronbestanden van programma's geen bestandsnaamextensies die eindigen op de letter *o*. Als u dat wel doet, wordt het bronbestand overschreven door het objectbestand bij het compileren van programma's.

De gecompileerde bronbestanden hebben een binaire indeling en kunnen niet worden weergegeven of gewijzigd.

Gecompileerde objectbestanden worden in dezelfde directory gemaakt als de broncodebestanden. Dit is de standaardinstelling. De standaard-bestandsextensies voor broncode en gecompileerde code worden weergegeven in Tabel 3.1.

Tabel 3.1 Standaard-bestandsextensies voor programmabestanden met bron- en objectcode

Soort bestand	Extensies voor broncodebestand	Extensies voor gecompileerd bestand
Programmabestand	.PRG	.PRO
Gegenereerd formulierbestand	.WFM	.WFO
Gegenereerd querybestand	.QBE	.QBO
Gegenereerd menubestand	.MNU	.MNO
dBASE IV-formulierbestand	.FMT	.FMO
dBASE IV-rapportbestand	.FRG	.FRO
dBASE IV-labelbestand	.LBG	.LBO

De bestandstypen van dBASE IV worden niet gemaakt in dBASE voor Windows, maar worden ondersteund vanwege neerwaartse compatibiliteit.

U kunt programmabestanden met verschillende dBASE-commando's compileren. Met COMPILE maakt u programma's op basis van een of meer bestanden zonder dat de resulterende objectcode wordt uitgevoerd. Met de volgende commando's wordt een programma gecompileerd als onderdeel van de handeling van het commando:

- DO. Hiermee wordt een programma gecompileerd en vervolgens gestart als er geen compilatiefouten zijn opgetreden.
- SET PROCEDURE TO *<bestand>*. Hiermee wordt gezocht naar een bestaand programmabestand dat al is gecompileerd. Als er geen .PRO-bestand wordt gevonden, wordt een .PRG-bestand gecompileerd.

Als u in plaats van DO of SET PROCEDURE het commando COMPILE gebruikt om bestanden te compileren, heeft dat de volgende voordelen:

- Met COMPILE worden de opgegeven bestanden niet uitgevoerd of geopend
- Met COMPILE kunt u jokertekens gebruiken in bestandsnamen en kunt u niet-gerelateerde of gerelateerde bestanden compileren

Als u een programma compileert, worden de syntaxisfouten in het bronbestand automatisch opgespoord en verschijnt een foutmelding in het dialoogvenster **Programmamelding**. Als tijdens de compilatie een foutmelding verschijnt, kunt u daarop reageren door op een van de volgende knoppen te klikken:

- Met de knop **Annuleren** annuleert u de compilatie. Dit heeft hetzelfde resultaat als wanneer u op *Esc* drukt.
- Met de knop **Negeren** annuleert u de compilatie van het programma met de syntaxisfout. De resterende bestanden die overeenkomen met het gebruikte jokerteken in het bestandsfilter, worden echter wel gecompileerd.
- Met de knop **Corrigeren** opent u de broncode in een editor-venster. De invoegpositie wordt op de coderegel met de syntaxisfout geplaatst.
- Met de knop **Help** kunt u contextgebonden Help-informatie oproepen.

Verschillende versies van bron- en objectbestanden vergelijken

De instelling van de SET DEVELOPMENT-omgeving bepaalt of dBASE de aanmaakdatum en -tijd van het bronbestand en het objectbestand vergelijkt. Als SET DEVELOPMENT is ingesteld op ON, wordt de datum en tijd van programma-, formulier- of procedurebestanden altijd vergeleken met die van de corresponderende objectbestanden. Als u een programmabestand wijzigt en dit niet meteen compileert, wordt het bestand automatisch gecompileerd wanneer u het gewijzigde bestand probeert te starten.

De standaardinstelling van SET DEVELOPMENT is ON. Als u deze standaardinstelling wilt wijzigen, doet u het volgende:

1 Kies **Kenmerken | Bureaublad**. Het dialoogvenster **Kenmerken bureaublad** verschijnt.

2 Selecteer de tab **Programmering**.

3 Schakel het aankruisvakje **Compilatie** uit.

Als u SET DEVELOPMENT wilt instellen op OFF vanuit het commandovenster, typt u het volgende:

```
SET DEVELOPMENT OFF
```

Als SET DEVELOPMENT is ingesteld op OFF, worden geen tijd- en datumstempels vergeleken en worden bestaande, reeds gecompileerde programma's, formulieren of query's uitgevoerd. Als er geen objectbestand bestaat, wordt het bronbestand gecompileerd, ongeacht de instelling van SET DEVELOPMENT.

Terwijl u het programma ontwerpt, gebruikt u SET DEVELOPMENT ON om ervoor te zorgen dat u de laatst bewerkte versie van een programma gebruikt. Als u geen programma's ontwerpt, gebruikt u SET DEVELOPMENT OFF, zodat de uitvoering van programma's wordt versneld.

Programma's testen

Een programma testen betekent dat u elke programmamodule afzonderlijk start en uitprobeert om fouten en zwakke punten in het programma op te sporen. Als u een programma test, kunt u ervoor zorgen dat het programma afdoende foutafhandelingsroutines bevat voor het geval dat het niet de juiste invoer ontvangt. U

kunt er bijvoorbeeld voor zorgen dat een foutmelding verschijnt wanneer een gebruiker numerieke waarden invoert in een tekstveld.

Als u een dBASE-applicatie wilt testen, hebt u testgegevens nodig voor de tabellen die de applicatie gebruikt. Bovendien moet u weten welk gedeelte van het programma werkelijk wordt uitgevoerd als u het programma start. Hiertoe beschikt u over de volgende commando's: GENERATE en SET COVERAGE.

Fouten in een programma opsporen is een andere manier waarop u een programma kunt controleren op fouten en zwakke punten. Met de debugger van dBASE beschikt u over krachtige hulpmiddelen waarmee u de uitvoering van een programma stap voor stap kunt volgen. Zie Hoofdstuk 8 voor een beschrijving van de debugger.

Willekeurige records genereren

Als u een grote tabel nodig hebt voor het uitvoeren van zware testen op een programma, kunt de tabel met behulp van het commando GENERATE opvullen met willekeurige tekens en getallen. Als een tabel al records bevat, worden deze door GENERATE intact gelaten en wordt het opgegeven aantal records aan de tabel toegevoegd.

Opmerking U kunt geen gegevens genereren in memovelden, binaire velden of OLE-velden.

Als u willekeurige records wilt genereren vanuit het Navigator- of catalogusvenster, gaat u als volgt te werk:

1 Open de tabel waaraan u de willekeurige records wilt toevoegen.

2 Kies **Tabel | Tabelhulpmiddelen | Genereren** en typ in het dialoogvenster **Records genereren** het aantal records dat u wilt toevoegen. Druk vervolgens op *Enter*.

Als u willekeurige records wilt genereren vanuit het commandovenster, doet u het volgende:

```
USE <tabelnaam>        && <tabelnaam> is de tabel waarvoor u records wilt genereren
GENERATE <uitdN>        && <uitdN> is het aantal records dat u wilt toevoegen
```

Dekkingsanalyse gebruiken

Met het commando SET COVERAGE kunt u een dekkingsanalyse uitvoeren voor programma's die zijn gecompileerd terwijl SET COVERAGE is ingesteld op ON. U kunt de resultaten van de dekkingsanalyse weergeven als u wilt controleren of alle modulen van uw programma worden uitgevoerd. Als sommige delen van uw programma niet worden uitgevoerd, markeert u deze delen, zodat u ze kunt testen en ervoor kunt zorgen dat ze wel functioneren. Als een subroutine bijvoorbeeld niet wordt aangeroepen omdat de voorwaarde die de subroutine aanroept niet van kracht is, kunt u de invoergegevens wijzigen zodat de voorwaarde wel van kracht is. Vervolgens kunt u controleren of de subroutine op de juiste wijze functioneert.

Als SET COVERAGE is ingesteld op ON en u een programma compileert en start, wordt het programma tijdens runtime geanalyseerd en wordt een dekkingsbestand

gemaakt. Dit is een binair bestand waarin de resultaten van de analyse worden opgeslagen. Het dekkingsbestand heeft dezelfde naam als het programma dat u analyseert en heeft bovendien de extensie .COV.

Als SET COVERAGE is ingesteld op ON, wordt door programma's die u compileert een ander .PRO-bestand gemaakt dan wanneer SET COVERAGE is ingesteld op OFF. Deze .PRO-bestanden bevatten informatie voor het genereren van dekkingsbestanden, ongeacht de instelling van SET COVERAGE tijdens runtime. De enige manier waarop u kunt voorkomen dat dekkingsbestanden worden gegenereerd, is door de programma's opnieuw te compileren terwijl SET COVERAGE is ingesteld op OFF, zodat .PRO-bestanden zonder dekkingsinformatie worden gegenereerd.

Dekkingsanalyse starten

Als u dekkingsanalyse wilt starten, stelt u SET COVERAGE in op ON en voert u het programma uit met de commando's COMPILE of DO. Als u de dekkingsanalyse wilt inschakelen, voert u een van de volgende stappen uit:

- Typ SET COVERAGE ON in het commandovenster en druk op *Enter*.
- Kies **Kenmerken | Bureaublad** of typ SET in het commandovenster. Schakel vervolgens het aankruisvakje **Dekking** in het tabblad **Programmering** van het dialoogvenster **Kenmerken bureaublad** in.
- Typ #pragma COVERAGE(ON) in een programmabestand.
- Typ het commando SET COVERAGE ON in een programmabestand.

Nadat u een van deze stappen hebt uitgevoerd, compileert en start u het programma om een dekkingsbestand voor het programma te maken. Tijdens de compilatie dient COVERAGE ingeschakeld te zijn. Als u COVERAGE uitschakelt tijdens het starten van het programma en wanneer het programma actief is, worden er toch dekkingsbestanden gemaakt.

Als u het commando SET COVERAGE ON in een programmabestand typt zonder COVERAGE in te schakelen voordat u begint met compileren, wordt geen dekkingsbestand voor dat programma gemaakt. Subroutines die door het programma worden aangeroepen terwijl COVERAGE is ingeschakeld, worden echter wel gecompileerd.

Als u bijvoorbeeld het programma MIJNCOMP.PRG start met de aangegeven reeks commando's, wordt de dekkingsanalyse automatisch ingeschakeld, worden het hoofdprogramma (HOOFD.PRG) en de bijbehorende programmabestanden met subroutines (SUB1.PRG and SUB2.PRG) gecompileerd en worden dekkingsbestanden (HOOFD.COV, SUB1.COV en SUB2.COV) gegenereerd voor deze drie programmabestanden:

```
* MijnComp.prg
#include "versie.h"  && Stelt Debug 1 in
#ifdef Debu
   SET COVERAGE ON
#endif
Compile Hoofd
```

```
Compile Sub1
Compile Sub2
```

In een programma met veel procedures wordt voor elke module een afzonderlijk dekkingsbestand gemaakt.

#pragma COVERAGE(ON) te gebruiken. Een programma waarvoor u COVERAGE inschakelt met #pragma, mag in de afsluitingsprocedure *geen* instructie #pragma COVERAGE(OFF) bevatten. Als u beide instructies in hetzelfde programma gebruikt, werkt alleen de tweede instructie.

#pragma COVERAGE(ON) in een programmabestand gebruikt, hoeft u SET COVERAGE ON niet te gebruiken voordat u het programma compileert. Bestaande dekkingsbestanden worden telkens bijgewerkt wanneer u wijzigingen aanbrengt in het programma en het vervolgens opnieuw compileert.

Voordat een dekkingsbestand kan worden gemaakt voor een programma, moet u dat programma eerst compileren met SET COVERAGE ON. Als u een programma compileert terwijl SET COVERAGE is ingesteld op OFF en u vervolgens SET COVERAGE instelt op ON en het programma start, verschijnt een foutmelding en wordt geen dekkingsbestand gemaakt.

Dekkingsanalyse beëindigen

Als u dekkingsanalyse wilt beëindigen, stelt u SET COVERAGE in op OFF en compileert u het programma opnieuw.

Als u SET COVERAGE wilt instelleń op OFF, voert u een van de volgende stappen uit:

- Typ SET COVERAGE OFF in het commandovenster en druk op *Enter*
- Schakel het aankruisvakje **Dekking** in het dialoogvenster **Kenmerken bureaublad** uit

Compileer het programma opnieuw terwijl COVERAGE is uitgeschakeld als u de dekkingsanalyse wilt beëindigen.

Bewerk de programma's met de preprocessor-instructie #pragma COVERAGE(ON) en wijzig de instructie als volgt:

```
#pragma COVERAGE (OFF)
```

Deze opdracht zorgt ervoor dat dekkingsanalyse is uitgeschakeld wanneer u het programma de volgende keer compileert.

Dekkingsbestanden maken vs. bijwerken

Informatie in een dekkingsbestand wordt telkens bijgewerkt wanneer u het programma compileert terwijl COVERAGE is ingeschakeld. Als u programma's compileert met SET COVERAGE ON en u de programmabestanden niet hebt gewijzigd, worden eventuele dekkingsbestanden voor die programma's bijgewerkt, ongeacht de instelling van SET COVERAGE tijdens het starten van de programma's in de testfase.

Als u wel een programmabestand wijzigt, opnieuw compileert en het programma start (waarbij SET COVERAGE is ingesteld op ON), wordt een nieuw dekkingsbestand gemaakt voor dat programma.

#pragma COVERAGE(ON), wordt een bestaand dekkingsbestand bijgewerkt. Als u wijzigingen aanbrengt in het programma dat de instructie #pragma COVERAGE(ON) bevat, moet u het programma opnieuw compileren en het vervolgens starten.

Logische blokken

Een dekkingsbestand is een binair bestand dat cumulatieve informatie bevat over het aantal keren dat de *logische blokken* van een programma in dBASE worden gestart en afgesloten (en dus volledig worden uitgevoerd). Een logisch blok bevat geen commentaarregels of *programmaverloop-opdrachtregels* zoals IF en ENDIF. Logische blokken bevatten echter wel opdrachtregels *binnen* programmaverloop-opdrachtregels.

Als uw programma geen programmaverloop-commando's (IF...ENDIF, DO WHILE...ENDDO, FOR...NEXT, SCAN...ENDSCAN, DO CASE...ENDCASE, DO...UNTIL) bevat, heeft het programma maar één logisch blok, dat bestaat uit alle commandoregels (met uitzondering van commentaarregels).

Het dekkingsbestand duidt een logisch blok aan met de bijbehorende nummers van de commandoregels. De commentaarregels in het volgende voorbeeld geven bijvoorbeeld aan welke regels in de dekkingsanalyse worden aangeduid als logische blokken. (Dit is niet de uitvoer van een dekkingsbestand.)

```
* BLOKBEP.PRG
SET TALK OFF                          && Regel 2,  Blok 1 (Regels 2-3)
USE Klanten INDEX Verkoper
SCAN
   DO CASE
      CASE Verkoper = "S-12"
         SELECT 2                     && Regel 7,  Blok 2 (Regels 7-8)
         USE S12
      CASE Verkoper = "L-5"
         SELECT 2                     && Regel 10, Blok 3 (Regels 10-11)
         USE L5
      CASE Verkoper = "J-25"
         SELECT 2                     && Regel 13, Blok 4 (Regels 13-14)
         USE J25
   ENDCASE
   DO Wijzig                          && Regel 16, Blok 5 (Regels 16-17)
   SELECT 1
ENDSCAN
CLOSE ALL                             && Regel 19, Blok 6 (Regels 19-20)
SET TALK ON
```

Informatie dekkingsbestand weergeven

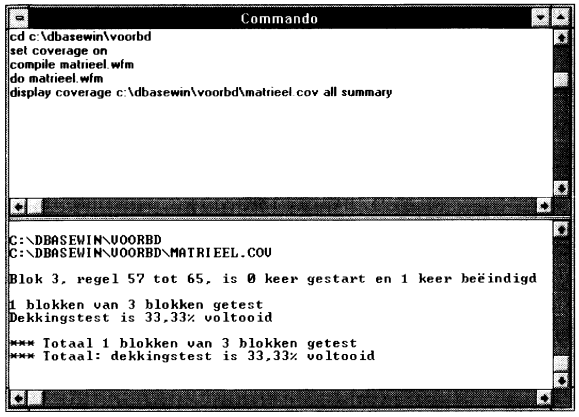
Gebruik DISPLAY COVERAGE als u de informatie scherm voor scherm wilt weergeven in het resultatenpaneel van het commandovenster.

Als de informatie die met DISPLAY COVERAGE of LIST COVERAGE wordt weergegeven, groter is dan het weergave-buffergeheugen, kunt u niet teruggaan naar het begin van de informatie zodra deze het resultatenpaneel van het commandovenster

is gepasseerd. Als u de informatie opnieuw wilt weergeven, typt u opnieuw DISPLAY COVERAGE of LIST COVERAGE.

Gebruik LIST COVERAGE als u de informatie wilt opslaan op diskette of deze wilt afdrukken. Zie de opties TO FILE en TO PRINTER bij de commando's DISPLAY COVERAGE en LIST COVERAGE in *Commando's en functies*.

Afbeelding 3.1 Uitvoer van DISPLAY COVERAGE in het resultatenpaneel van het commandovenster



Hoofdstuk

4

Werken met procedures en codeblokken

Bij modulair programmeren worden programmeertaken verdeeld in kleine, op zichzelf staande programma-eenheden, ofwel modulen, die eenvoudig kunnen worden bijgewerkt en opnieuw gebruikt. In modulen kunnen fouten ook gemakkelijker worden opgespoord. Deze modulen kunnen procedures (of functies) en codeblokken zijn. Bovendien kunnen deze modulen met elkaar of met uw hoofdprogramma samenwerken om gecompliceerde taken te verrichten.

Evenals in dBASE IV kunt u in dBASE voor Windows programmamodulen definiëren als procedures of als functies. Bovendien ondersteunt dBASE voor Windows ook *codeblokken*. Codeblokken zijn kleine, naamloze eenheden met code die u toewijst aan variabelen of die u rechtstreeks in andere opdrachten insluit.

Omdat in dBASE geen onderscheid wordt gemaakt tussen procedures en functies, wordt in dit hoofdstuk met de term *procedure* verwezen naar routines die worden gedefinieerd met PROCEDURE of met FUNCTION.

In dit hoofdstuk wordt een inleiding gegeven tot procedures en codeblokken, worden de verschillen tussen beide uitgelegd en wordt beschreven hoe u procedures en codeblokken maakt. Daarnaast wordt aangegeven hoe u procedures kunt aanroepen en hoe u codeblokken in programma's gebruikt.

Over procedures en codeblokken

dBASE ondersteunt procedures, functies, codeblokken met opdrachten en codeblokken met uitdrukkingen.

- Een *procedure* is een kort programma dat een of meer programma-opdrachten bevat en een taak uitvoert. Een procedure kan een waarde als resultaat geven.

- Een *functie* is een kort programma dat lijkt op een procedure. Een functie moet echter een resultaatwaarde bevatten.
- *Codeblokken met opdrachten* zijn logisch verbonden groepen programma-opdrachten die geen waarde als resultaat geven.
- *Codeblokken met uitdrukkingen* zijn groepen uitdrukkingen waartussen een relatie bestaat en die altijd een waarde als resultaat geven.

Met codeblokken kunt u routines die worden gebruikt voor het verrichten van een bepaalde taak, verbergen voor de rest van de code. U kunt procedures of codeblokken toewijzen aan geheugenvariabelen of aan een objectkenmerk.

In Tabel 4.1 wordt een overzicht gegeven van het gebruik van procedures, functies en codeblokken.

Tabel 4.1 Overzicht van het gebruik van procedures, functies en codeblokken

	Gebruik	**Voorbeeld**
Procedures en functies	Voor het maken van een benoemde, opnieuw te gebruiken groep programma-opdrachten waarmee een taak kan worden uitgevoerd en die een waarde als resultaat kunnen geven.	Een actie-afhandelingsroutine OnClick voor een knop waarmee een waarde wordt berekend; een actie-afhandelingsroutine Valid voor een invoervak.
Codeblokken met opdrachten	Voor het maken van een subroutine die bestaat uit één of een aantal commando's die kunnen worden toegewezen aan een geheugenvariabele of een objectkenmerk.	Een actie-afhandelingsroutine OnClick waarmee wordt doorgegaan naar de volgende record.
Codeblokken met uitdrukkingen	Voor het maken van een subroutine die bestaat uit één uitdrukking, die tijdens runtime een dynamisch geëvalueerde waarde als resultaat geeft.	Een actie-afhandelingsroutine Valid of When.

Procedures vs. functies

In dBASE IV zijn er grote verschillen tussen procedures en functies. Alleen functies kunnen waarden als resultaat geven, en alleen procedures kunnen worden aangeroepen met het commando DO. Deze beperkingen gelden niet in dBASE.

Bij functies moet u een uitdrukking opnemen in de opdracht RETURN. Bij procedures kunt u een dergelijke uitdrukking echter ook gebruiken. U kunt zowel procedures als functies aanroepen met een opdracht DO, zoals in DO MijnProc, of met de aanroepingsoperator (haakjes), zoals in MijnProc(). Als u een procedure aanroept als een functie met behulp van de aanroepingsoperator, moet deze procedure een waarde als resultaat geven.

Als u een actie-afhandelingsroutine schrijft met Formulierontwerp, wordt geen functie maar een procedure gedefinieerd, zelfs wanneer de actie-afhandelingsroutine een waarde als resultaat geeft.

Als u gewoon bent procedures en functies voor verschillende doeleinden te definiëren, kunt u dat blijven doen. Omdat procedures en functies uitwisselbaar zijn in dBASE voor Windows, worden alleen procedures gebruikt in de codevoorbeelden in dit hoofdstuk.

Procedures definiëren

Als u een procedure in een programma wilt definiëren, moet u de procedure eerst omzetten in code. Vervolgens moet u de procedure compileren om te zien of deze geen fouten bevat. Daarna moet u de procedure toevoegen aan het einde van het programmabestand. U kunt ook een apart procedurebestand maken dat veelgebruikte subroutines bevat.

De eerste regel van een procedure luidt: PROCEDURE <*NaamProcedure*>. Vervolgens kunt u de opdrachten schrijven waarmee de taak wordt uitgevoerd. De laatste regel van de procedure luidt: RETURN.

Zie *Commando's en functies* voor een volledige beschrijving van het commando RETURN.

Plaats procedures aan het einde van het programmabestand waarin u de desbetreffende procedure(s) wilt aanroepen.

In het volgende voorbeeld wordt een procedure gedefinieerd waarmee u record voor record door een tabel kunt schuiven. Als u deze opdrachten niet telkens in uw programma wilt herhalen wanneer u door een tabel wilt schuiven, gebruikt u de opdracht DO RecOversl of koppelt u de procedure aan de actie OnClick van een knop Volgende record.

```
PROCEDURE RecOversl
   SKIP          && Naar volgende record
   IF EOF()      && Als bestandseinde
      GO TOP     && Naar eerste record in tabel
   ENDIF
RETURN
```

Parameters definiëren

Met parameters voegt u verschillende waarden toe aan een procedure en wijzigt op die manier de werking van de procedure. De resulterende waarde van een procedure is afhankelijk van de waarden die u opgeeft. Het doorgeven van parameters is een veelgebruikte manier om waarden naar een procedure te sturen en waarden terug te sturen naar het aanroepende programma.

Parameters zijn een of meer geheugenvariabelen die u in een proceduredefinitie opneemt en waarmee waarden aan de procedure worden doorgegeven. Met parameters wijzigt u de werking van een procedure of wisselt u waarden uit met andere programmamodulen.

U definieert parameters in een procedure door de parameters tussen haakjes te plaatsen achter de procedurenaam. Als u haakjes gebruikt, worden de parameters lokaal gebruikt in het bereik van de procedure. LOCAL-geheugenvariabelen kunnen niet worden gewijzigd door subroutines die na de variabelen worden aangeroepen. Zie Hoofdstuk 5 voor meer informatie over het bereik van geheugenvariabelen.

In dBASE IV worden parameters gedefinieerd met een instructie PARAMETERS in de procedure. Deze methode voor de definitie van parameters wordt hoofdzakelijk

ondersteund vanwege compatibiliteit met dBASE IV. Deze methode wordt echter niet aanbevolen, aangezien u hiermee PRIVATE-geheugenvariabelen maakt. PRIVATE-variabelen kunnen per ongeluk worden overschreven door een andere procedure die dezelfde namen van variabelen gebruikt.

De volgende twee definities van klassen zijn identiek, met dien verstande dat in de ene een parameter met haakjes wordt gedefinieerd en in de andere met PARAMETERS.

Afbeelding 4.1 Parameters definiëren met een LOCAL en een PRIVATE bereik

Een LOCAL-parameter definiëren

```
PROCEDURE Kwadraat(n)
   Getal = n *n
RETURN
```

Een PRIVATE-parameter definiëren

```
PROCEDURE Kwadraat
   PARAMETERS n
   Getal = n*n
RETURN
```

In het volgende voorbeeld wordt de procedure uit het voorbeeld met RecOversl gewijzigd, zodat een variabele kan worden opgegeven voor het aantal records dat moet worden overgeslagen en de recordaanwijzer met hetzelfde aantal wordt verplaatst. U roept deze procedure aan met een van de eerste twee opdrachten uit het voorbeeld.

```
DO RecOversl WITH 10            && 10 records overslaan
RecOversl(10)                   && Equivalent aan vorige opdracht

PROCEDURE RecOversl(AantRec)    && Proceduredefinitie met parameter AantRec
                                && AantRec geeft waarde van ThisMany
IF TYPE("AantRec") = "N"        && Variabele controleren op type Numeriek
   SKIP AantRec                 && Aantal in AantRec overslaan
   IF EOF()                     && Als bestandseinde
      GO TOP                    && Naar eerste record in tabel
   ENDIF
ENDIF
RETURN
```

Als een procedure geen waarde als resultaat geeft, kunt u de opdracht RETURN aan het einde van de procedure weglaten. Een procedure is afgelopen als alle invoer die u hebt opgegeven, wordt verwerkt en als de programma-uitvoering doorgaat met de volgende coderegel in het programma dat de procedure heeft aangeroepen. Met een opdracht RETURN zonder argumenten kunt u de programmacontrole teruggeven aan de module die de procedure heeft aangeroepen, of aan een andere procedure. Hiertoe geeft u de naam van de procedure op bij de opdracht RETURN. Zie RETURN in *Commando's en functies* voor de beschikbare opties.

Procedures aanroepen met parameters

In deze paragraaf worden voorbeelden gegeven van het doorgeven van parameters aan procedures en van de wijze waarop resultaatwaarden worden verkregen. U kunt parameters op twee manieren aan een procedure doorgeven:

- Gebruik de aanroepingsoperator (haakjes) achter de procedurenaam en plaats de parameter tussen de haakjes
- Gebruik de syntaxis DO *<Procedurenaam>* WITH

In het volgende voorbeeld worden beide technieken voor het doorgeven van parameters toegelicht.

```
Getal1 = 3
Getal2 = 5
Getal3 = 10
Antwoord1 = MijnBerekening(Getal1,Getal2,Getal3)
? Antwoord1                                    && Resultaat is 80 ((3+5)*10)
Totaal = 0
DO MijnBerekening WITH 4,8,9
? Totaal                                       && Resultaat is 108 ((4+8)*9)

PROCEDURE MijnBerekening(Eerste, Tweede, Derde) && Proceduredefinitie
                                               && en parameteropdracht.
Totaal = (Eerste + Tweede) * Derde             && Uitdrukking die de parameters evalueert
RETURN Totaal                                  && Resultaatwaarde
```

Als u een procedure aanroept met DO, wordt geen waarde als resultaat gegeven, zelfs niet als u een waarde hebt opgegeven in de opdracht RETURN. In plaats daarvan moet u parameters en geheugenvariabelen gebruiken als u wilt dat de procedure waarden als resultaat geeft die u nodig hebt. In het volgende voorbeeld wordt dit toegelicht.

```
Waarde = 12
? Kwadraat(Waarde)              && Resultaat is 144
DO Kwadraat WITH Waarde
? Waarde                        && Resultaat is 12
DO Kwadraat1 WITH Waarde        && Kwadraat1 werkt de geheugenvariabele bij die als
                                && parameter wordt doorgegeven
? Waarde                        && Resultaat is 144

PROCEDURE Kwadraat(mWaarde)     && Dit is de procedure Kwadraat, waarbij u één parameter
                                && kunt gebruiken
Kw_waarde = mWaarde*mWaarde     && Evalueert deze uitdrukking
RETURN Kw_waarde                && Geeft een waarde als resultaat

PROCEDURE Kwadraat1(mWaarde)    && Dit is de procedure Kwadraat1
mWaarde = mWaarde*mWaarde       && Evalueert deze uitdrukking
RETURN
```

In het volgende voorbeeld ontvangt de procedure NaarReeks een parameter met het gegevenstype Numeriek, Logisch of Teken. NaarReeks converteert de parameter zo nodig naar een reeks, breekt de reeks af, voegt een spatie toe en geeft een waarde als resultaat.

```
PROCEDURE NaarReeks(GeenReeks)
LOCAL EenReeks
DO CASE
   CASE TYPE("GeenReeks")$"NI"          && Numeriek of zwevend decimaalteken
      EenReeks = STR(GeenReeks,2)
   CASE TYPE("GeenReeks") = "L"         && Logisch
      IF GeenReeks
          EenReeks = "True"
      ELSE
          EenReeks = "False"
      ENDIF
   CASE TYPE("GeenReeks") = "T"         && Teken
   EenReeks = GeenReeks
   OTHERWISE                            && Geen van de bovenstaande
      EenReeks = "?"
ENDCASE
EenReeks = LTRIM(RTRIM(EenReeks))
EenReeks = EenReeks+" "
RETURN EenReeks
```

Een procedure zoals NaarReeks is handig als u een willekeurige variabele wilt doorgeven en een opgemaakt resultaat wilt terugkrijgen.

```
Bedrijf = "Janssen & Janssens          "
Hoeveelheid = "5000"
AllesGoed= .T.
? NaarReeks(Bedrijf) + NaarReeks(Hoeveelheid) + NaarReeks(AllesGoed)
* Resultaatreeks: Janssen en Janssens 5000 .T.
```

In het volgende voorbeeld ziet u hoe u een array doorgeeft aan een procedure. Geef de naam van de array door als u de array wilt doorgeven. In het voorbeeld wordt het gemiddelde berekend van alle getallen die aan de array worden doorgegeven.

```
DECLARE eenGetal[5]
FOR i=1 TO 5
   eenGetal[i]=i                  && Array opvullen met de waarden 1,2,3,4 en 5
NEXT i
GetalGem=ArrayGem(eenGet)         && Array eenGet doorgeven
? GetalGem                        && Resultaat is 3 ((1+2+3+4+5)/5)

PROCEDURE ArrayGem(X)             && Proceduredefinitie met één parameter: X
Sum=0                             && Variabele initialiseren op nul
FOR i = 1 TO ALEN(X)              && Resultaat van ALEN() is aantal elementen in array
   Sum = Sum+X[i]
NEXT i
RETURN Sum/ALEN(X)
```

Array-variabelen kunnen ook afzonderlijk worden doorgegeven, zoals u in het volgende voorbeeld kunt zien:

```
DO MijnProc WITH X[i],X[i+1]
```

Zie Hoofdstuk 5 voor meer informatie over array's.

Het aantal parameters variëren

Het aantal parameters dat wordt doorgegeven, kan groter of kleiner zijn dan het aantal dat in de procedure wordt gedefinieerd.

- Als u minder parameters doorgeeft dan u hebt gedefinieerd, worden verwachte maar niet doorgegeven parameters ingesteld op .F.
- Als u meer parameters doorgeeft dan u hebt gedefinieerd, worden de extra parameters genegeerd

Met PCOUNT() kunt u bepalen hoeveel parameters zijn doorgegeven. In het volgende voorbeeld ziet u hoe u PCOUNT() gebruikt:

```
DO MijnProc WITH A,B,C                  && Drie parameters doorgeven
DO MijnProc WITH A,B,C,D,E              && Vijf parameters doorgeven

PROCEDURE MijnProc(P1,P2,P3,P4)         && Verwacht vier parameters
IF PCOUNT() >= 4
   * Opgegeven taken uitvoeren als voldoende parameters zijn doorgegeven
ELSE
   * Melding weergeven
ENDIF
RETURN
```

In het volgende programma wordt gebruik gemaakt van de mogelijkheid om het aantal parameters te variëren. Als geen parameter wordt doorgegeven, wordt gedurende twee seconden een venster weergegeven. U kunt deze procedure aanroepen met een parameter als u de wachttijd wilt wijzigen.

```
PROCEDURE WachtVen (Timeout)
Melding = "Druk op een toets om door te gaan"
DO CASE
   CASE PCOUNT() = 0            && Geen timeout-waarde doorgegeven
      Timeout = 2               && Timeout-waarde instellen op 2 seconden
   CASE PCOUNT() > 1            && Melding wijzigen als te veel parameters doorgegeven
      Melding = "Waarschuwing, te veel parameters. " + Msg
ENDCASE
Lengte = (LEN(Msg) + 8) / 2     && Grootte van te maken formulier
DEFINE FORM Wachten;
   FROM 10,40 - Lengte TO 14,40 + Lengte
DEFINE TEXT WachtTekst OF Wachten;
   AT 1,2;
   PROPERTY ;
      Tekst Melding             && Melding in formulier
Wachten.Open()                  && Formulier openen
Tewachtentijd = INKEY(Timeout)  && Aantal seconden van Timeout of op toetsaanslag wachten
Wachten.Close()                 && Formulier sluiten
RETURN
```

Dit zijn enkele voorbeelden van aanroepen voor WachtVen:

```
WachtVen()                      && 2 seconden wachten
WachtVen(5)                     && 5 seconden wachten
WachtVen(3,5)                   && 3 seconden wachten en waarschuwingsmelding weergeven
```

Namen van geheugenvariabelen beschermen

De waarden van bestaande geheugenvariabelen kunnen door alle procedures en functies worden gewijzigd. Zie Hoofdstuk 5 voor een beschrijving van het type en het bereik van geheugenvariabelen. In veel gevallen wilt u waarschijnlijk dat de procedures de huidige variabelen gebruiken om de gewenste waarden als resultaat te geven. In sommige gevallen wilt u de huidige waarden van geheugenvariabelen misschien beschermen. U kunt bepalen wat er met de waarden van geheugenvariabelen gebeurt door parameters door te geven *als verwijzing* of *als waarde*.

- Met *doorgeven als verwijzing* geeft u een verwijzing door aan de variabele. Eventuele wijzigingen die door de procedure worden aangebracht in de variabele, worden weergegeven in de waarde van de variabele wanneer de procedure is afgelopen.
- Met *doorgeven als waarde* geeft u de waarde van de variabele door, maar niet de variabele zelf. Als u parameters doorgeeft als waarde, wordt de waarde van de variabele door de ontvangende subroutine opgeslagen in een nieuwe variabele en wordt de oorspronkelijke variabele niet gewijzigd. Als u een parameter wilt doorgeven als waarde, plaatst u haakjes rond de variabele(n) die u wilt doorgeven.

In het volgende voorbeeld ziet u de verschillen tussen de twee manieren waarop u parameters kunt doorgeven. De haakjes zorgen ervoor dat dBASE de waarde van X evalueert als een uitdrukking en het resultaat van die uitdrukking doorgeeft.

```
x = 10
? x                     && x = 10
DO MijnProc WITH x      && als verwijzing doorgeven, procedure kan waarde van x wijzigen
? x                     && x = 11 (x-1, gewijzigd door MijnProc)
DO MijnProc WITH (x)    && als waarde doorgeven; procedure kan waarde van x gebruiken,
                        && maar niet wijzigen
? x                     && x = 11 (niet gewijzigd door MijnProc)

PROCEDURE MijnProc(n)
? n                     && 10 weergeven bij eerste aanroep, 11 bij tweede
n = n+1
? n                     && 11 weergeven bij eerste aanroep, 12 bij tweede
?
RETURN
```

Procedurebestanden en -bibliotheken gebruiken

Tijdens runtime zoekt dBASE in bepaalde bestanden naar procedures. Deze bestanden worden hierna weergegeven in de volgorde waarin ze worden doorzocht.

1 Het uitvoerende objectbestand (met de extensie .PRO) van het programma.

2 Andere geopende objectbestanden (met de extensie .PRO) in de aanroepingsketen, in de volgorde waarin de bestanden het meest recent zijn geopend.

3 Het bestand dat is opgegeven bij SYSPROC = <*bestandsnaam*> in DBASEWIN.INI.

4 Alle bestanden die zijn geopend met de opdrachten SET PROCEDURE, SET PROCEDURE...ADDITIVE of SET LIBRARY, in de volgorde waarin de bestanden zijn geopend.

5 Het objectbestand (met de extensie .PRO) met de opgegeven naam in het zoekpad.

6 Het programmabestand (met de extensie .PRG) met de opgegeven naam in het zoekpad. Dit bestand wordt in dBASE automatisch gecompileerd.

Als meerdere procedures dezelfde naam hebben, wordt de eerste procedure uitgevoerd die wordt aangetroffen.

Voor optimale prestaties dient u het zoekpad voor de procedures die u gebruikt, zo kort mogelijk te houden. Zorg dat gecompileerde procedures beschikbaar zijn voor programma's door de procedures in procedurebibliotheken of in het programma zelf op te nemen.

Procedures opslaan in programmabestanden

Een programmabestand kan maximaal 193 procedures bevatten. U kunt echter een groot aantal procedurebestanden openen, die elk maximaal 193 procedures en een extra bibliotheekbestand kunnen bevatten. Het aantal procedures dat u kunt benaderen, wordt derhalve alleen beperkt door het beschikbare geheugen.

Het gebruik van procedures in een programma is zinvol als bepaalde procedures alleen worden aangeroepen door een bepaald programma. Als u het programma en de aangeroepen procedures in hetzelfde bestand opneemt, kunt u het bestand gemakkelijker bijwerken. Bovendien is het mogelijk dat de prestaties dan enigszins worden verbeterd, aangezien geen extra bestanden met procedures hoeven te worden geopend.

Procedures die zijn opgenomen in een programma, zijn beschikbaar zolang het programma actief is.

Programmabestanden die één procedure bevatten

U kunt één procedure schrijven in een programmabestand (.PRG-bestand) dat dezelfde naam heeft. In dat geval kunt u het commando FUNCTION of PROCEDURE weglaten. Over het algemeen gebruikt u deze techniek alleen voor het testen van een procedure en het opsporen van fouten daarin. In het volgende voorbeeld is MIJNPROG.PRG een procedure met de naam MijnProg die u kunt aanroepen met de opdracht

```
DO MijnProg WITH mvar1, mvar2.
```

```
* MijnProg.prg
PARAMETER x,y          && Opdracht PARAMETER nodig omdat opdracht PROCEDURE ontbreekt
x = x + y              && Notatie PPROCEDURE MijnProg(x,y) kan dus niet worden gebruikt
RETURN
* end of MijnProg.prg
```

Programmabestanden die meerdere procedures bevatten

Een programma kan meerdere procedures bevatten die door het programma (of de bijbehorende subroutines en procedures) worden aangeroepen. Dit wordt in het volgende voorbeeld toegelicht.

```
*MijnProgs.prg
DO MijnProc
DO Prog1                    && Prog1.prg is afzonderlijk bestand op schijf
                            && waarmee ook MijnProc of MijnAndereProc kan worden aangeroepen
DueDate = MijnAndereProc()
RETURN

PROCEDURE MijnProc
:
RETURN

PROCEDURE MijnAndereProc
Mvar = DATE() + 30
RETURN Mvar
* End of MijnProgs.prg
```

Procedures opslaan in procedurebestanden

U kunt procedurebestanden maken als u groepen wilt samenstellen van procedures die u in meerdere programma's gebruikt. Als u procedures op deze wijze opslaat, zijn groepen procedures beschikbaar voor elk gewenst programma. Veel programmeurs maken een set basisprocedures die zij opslaan in een of meer procedurebestanden. U kunt afzonderlijke procedurebestanden maken voor verschillende soorten procedures.

Als u een procedurebestand wilt openen, gebruikt u SET PROCEDURE TO <*bestandsnaam*>. Als u extra procedurebestanden wilt openen terwijl al een procedurebestand is geopend, gebruikt u de optie ADDITIVE bij SET PROCEDURE als volgt:

```
SET PROCEDURE TO <bestandsnaam2> ADDITIVE
```

Als u de optie ADDITIVE niet gebruikt, worden alle geopende procedurebestanden eerst gesloten voordat een ander procedurebestand wordt geopend.

```
SET PROCEDURE TO VideoBib              && Procedures in VideoBib beschikbaar
SET PROCEDURE TO GeluidsBib ADDITIVE   && Procedures in GeluidsBib ook beschikbaar
SET PROCEDURE TO GeluidsBib            && Procedures in VideoBib niet meer beschikbaar
```

Als u alle geopende procedurebestanden wilt sluiten, gebruikt u SET PROCEDURE TO zonder opties. Als u een bepaald procedurebestand wilt sluiten, gebruikt u CLOSE PROCEDURE <*bestandsnaam*>.

Procedures opslaan in bibliotheekbestanden

Naast procedurebestanden ondersteunt dBASE voor Windows ook bibliotheekbestanden, hoofdzakelijk vanwege compatibiliteit met dBASE IV. Procedures worden eerst gezocht in procedurebestanden en vervolgens in

bibliotheekbestanden. In dBASE voor Windows worden alle geopende procedure- en bibliotheekbestanden doorzocht in de volgorde waarin deze zijn geopend. Het belangrijkste verschil tussen procedure- en bibliotheekbestanden is dat u maar één bibliotheekbestand tegelijk geopend kunt hebben. Hierdoor zijn er geen voordelen verbonden aan het gebruik van bibliotheekbestanden in dBASE voor Windows. Het gebruik van procedurebestanden wordt daarom aanbevolen.

Functie-aanwijzers en codeblokken gebruiken

In vorige versies van dBASE zijn gegevenstypen alleen van toepassing op de gegevens waarmee gebruikers kunnen werken, zoals namen, hoeveelheden en datums. In dBASE voor Windows zijn gegevenstypen ook van toepassing op bepaalde gegevens waarmee programmeurs werken, zoals objecten, procedures of codeblokken. Het werken met blokken code alsof het blokken gegevens zijn, is een essentieel onderdeel van object-georiënteerd programmeren.

Voor het werken met code zijn de volgende gegevenstypen aan dBASE voor Windows toegevoegd:

- *Functie-aanwijzer* is een nieuw gegevenstype voor het opslaan van verwijzingen naar procedures of functies. Een variabele van het type functie-aanwijzer bevat een verwijzing naar een procedure of functie die u kunt toewijzen aan een geheugenvariabele of kenmerk, doorgeven als een parameter of resultaatwaarde, of rechtstreeks aanroepen.
- *Codeblok* is een nieuw gegevenstype voor het opslaan van kleine, naamloze groepen commando's of uitdrukkingen. Codeblokken kunt u gebruiken in uitdrukkingen, toewijzen aan geheugenvariabelen of objectkenmerken, doorgeven als parameters of resultaatwaarden, of rechtstreeks aanroepen.

De syntaxis en regels voor het schrijven van een codeblok met *opdrachten* zijn als volgt:

```
{|<parameters>|}; <opdracht> [; <opdracht> ...]}
```

- De accoladen, { }, zijn vereist
- Als u parameters doorgeeft, moeten deze tussen sluistekens (| |) worden geplaatst
- Laat elke opdracht voorafgaan door een puntkomma (;)

De syntaxis en regels voor het schrijven van een codeblok met een *uitdrukking* zijn als volgt:

```
{|<parameters>|} <uitdrukking>}
```

- De accoladen, { }, zijn vereist
- Als u parameters doorgeeft, moeten deze tussen sluistekens (| |) worden geplaatst

De regels voor het gebruik van gegevens van het type functie-aanwijzer en codeblok komen overeen. Het verschil is dat functie-aanwijzers naar code *verwijzen* en dat codeblokken code *bevatten*. In de rest van deze sectie worden voorbeelden gegeven.

Parameters

Als u parameters in een codeblok gebruikt, plaatst u deze aan het begin van het codeblok tussen sluistekens (| |). In het volgende voorbeeld wordt een codeblok met twee parameters, *a* en *b*, gemaakt:

```
GelijkAan = {|a,b|;? LTRIM(STR(a))+" plus "+LTRIM(STR(b))+" = "+LTRIM(STR(a+b))}
```

U roept het codeblok net als een functie aan, waarbij u de parameterwaarden tussen de haakjes van de aanroepingsoperator plaatst.

```
? GelijkAan(2,2)                    && Resultaat is "2 plus 2 = 4"
? GelijkAan(1,5)                    && Resultaat is "1 plus 5 = 6"
```

Dynamisch geëvalueerde uitdrukkingen

Als u een functie-aanwijzer gebruikt voor een codeblok met een uitdrukking, kunt u de expressie tijdens runtime dynamisch laten evalueren. In de volgende code worden waarden opgeslagen in geheugenvariabelen. Deze waarden worden eenmaal geëvalueerd op het moment dat de toewijzingsopdracht wordt uitgevoerd en blijven verder ongewijzigd, tenzij ze opnieuw worden toegewezen.

```
X = 5                               && 5 opslaan in X
Y = 6                               && 6 opslaan in Y
Z = X+ Y                            && 11 opslaan in Z
? Z                                 && Resultaat is 11
X = X+ 1                            && X is nu gelijk aan 6
? Z                                 && Resultaat is nog steeds 11
```

Vergelijk nu het voorgaande voorbeeld met het volgende, waarin een codeblok wordt gebruikt.

```
X = 5                               && 5 opslaan in X
Y = 6                               && 6 opslaan in Y
Z = { | Param1 , Param2 | Param1 + Param2 }  && Twee parameters definiëren in codeblok
? Z(X,Y)                            && Resultaat is 11
X = X+ 1                            && X is nu gelijk aan 6
? Z(X,Y)                            && Resultaat is 12
```

In het tweede voorbeeld bevat de variabele *z* een codeblok met een uitdrukking. Telkens wanneer *z* wordt uitgevoerd, wordt de uitdrukking geëvalueerd met behulp van de parameters die u doorgeeft. In dit geval zijn dat *x* en *y*.

U kunt de codeblokvariabele *z* als een parameter doorgeven aan andere functies of procedures, of u kunt *z* gebruiken als een dynamisch geëvalueerde variabele.

Actie-afhandelingsroutines

Codeblokken zijn bijzonder handig voor het toewijzen van code aan het kenmerk van een actie-afhandelingsroutine van een object, zoals u in de volgende voorbeelden kunt zien.

```
form.MijnKnop.Onclick = {;CLOSE FORM form}      && Codeblok met opdracht
form.InvoerHoevlh.Valid   = {this.Value > 0}    && Codeblok met uitdrukking
```

Als u codeblokken op deze wijze gebruikt, is uw code altijd overzichtelijk en kunt u eenvoudig bepalen welke handelingen aan een bepaald kenmerk zijn toegewezen.

In het volgende voorbeeld worden het formulier MijnForm en de knop VolgKnop gemaakt. De procedure RecOversl wordt toegewezen aan het kenmerk OnClick van VolgKnop, zodat een functie-aanwijzer wordt gemaakt.

```
* MijnFormDef.prg
MijnForm = NEW FORM()
VolgKnop = NEW PUSHBUTTON(MijnForm)
VolgKnop.OnClick = RecOversl                && Functie-aanwijzer maken

PROCEDURE RecOversl
   SKIP
   IF EOF()
      GO TOP
   ENDIF
RETURN
```

Belangrijk Een procedure moet beschikbaar zijn, ofwel in het huidige programmabestand ofwel in een geopend procedurebestand, wanneer u deze toewijst aan een variabele of kenmerk. De verwijzing naar de procedure wordt op het moment van toewijzing gerealiseerd.

In het volgende voorbeeld wordt een codeblok toegewezen aan de variabele *VolgRec*. Vervolgens wordt de variabele toegewezen aan een actiekenmerk.

```
VolgRec = {;SKIP;IF EOF();GO TOP;ENDIF}
VolgKnop.OnClick = VolgRec
```

Omdat dit codeblok zeer kort is, kunt u overwegen het codeblok aan de actie toe te wijzen, in plaats van het codeblok eerst aan een variabele toe te wijzen.

```
VolgKnop.OnClick = {;SKIP;IF EOF();GO TOP;ENDIF}
```

Een toegewezen codeblok is geïsoleerd en kan niet gemakkelijk op een andere plaats in een programma worden gebruikt. Als u een toegewezen codeblok opnieuw wilt gebruiken, moet u het kopiëren en het op elke plaats toewijzen waar u het wilt gebruiken. Procedures zijn meer geschikt voor het opstellen van code voor lange taken die vaak worden herhaald. Codeblokken zijn meer geschikt voor korte taken of eenmalige acties.

In het volgende voorbeeld wordt een codeblok met een volledige opdracht IF...ENDIF toegewezen aan een geheugenvariabele.

```
OpenOfMaak={;IF FILE("MijnBest.dbf"); USE MijnBest; ELSE; CREATE MijnBest FROM
BestStru;ENDIF}
```

In dit voorbeeld bevat OpenOfMaak een codeblok dat controleert of MIJNBEST.DBF aanwezig is, dat het bestand opent als het aanwezig is of dat het bestand maakt op basis van het bestand BESTSTRU. Als u het codeblok wilt starten, typt u OpenOfMaak() in een programma nadat u de geheugenvariabele hebt gemaakt.

In het volgende voorbeeld ziet u hoe u dezelfde taak in een procedure programmeert, in plaats van in een codeblok.

```
DO OpenOfMaak
*...
PROCEDURE OpenOfMaak
IF FILE("MIJNBEST.DBF")
   USE MijnBest
ELSE
   CREATE MijnBest FROM BestStru
ENDIF
```

Hoofdstuk

5

Werken met geheugenvariabelen

Een *geheugenvariabele*, of *variabele*, is een benoemde lokatie in het geheugen voor de tijdelijke opslag van gegevens. In een geheugenvariabele wordt één gegevenseenheid opgeslagen, die is te vergelijken met een veld in een tabel. Een *array* is een set geheugenvariabelen die zijn ingedeeld in rijen en kolommen, te vergelijken met records in een tabel. Programmeurs gebruiken geheugenvariabelen doorgaans voor de volgende activiteiten:

- Berekeningen uitvoeren en de resultaten daarvan tijdelijk opslaan
- Programma-uitvoering beheren met programmaverloop-commando's
- Constante waarden, zoals globale configuratie-instellingen, opslaan

In dit hoofdstuk worden geheugenvariabelen geïntroduceerd, wordt het bereik van geheugenvariabelen beschreven en worden technieken besproken voor het werken met geheugenvariabelen.

De volgende hoofdstukken bevatten eveneens onderwerpen over geheugenvariabelen:

- In Hoofdstuk 24 worden *systeemgeheugenvariabelen* beschreven die worden gebruikt voor het configureren van printers. (Hoofdstuk 5 van *Commando's en functies* bevat een volledige beschrijving van alle systeemgeheugenvariabelen.)
- In Hoofdstuk 18 worden commando's voor het maken van *automem*-variabelen beschreven. Met deze variabelen kunt u gegevens in velden snel ophalen en vervangen.
- In Hoofdstuk 10 worden *objectverwijzing*-variabelen beschreven. Deze variabelen verwijzen naar objecten.

Over geheugenvariabelen

Geheugenvariabelen zijn tijdelijke plaatsvervangers voor waarden. U maakt een geheugenvariabele door een waarde toe te wijzen aan een naam met het commando

STORE of met de gelijk-operator (=). Als u de waarde wilt ophalen, gebruikt u gewoon de naam van de variabele in een uitdrukking. Bij de evaluatie van de uitdrukking wordt de naam van de variabele vervangen door de waarde. Het volgende voorbeeld bevat enkele eenvoudige toewijzingen van geheugenvariabelen, die vervolgens in uitdrukkingen worden gebruikt.

```
? 5 * 3                  && Uitdrukking zonder variabelen (resultaat is 15)
STORE 5 TO mVar1         && Variabele mVar1 met waarde 5 maken
mVar2 = 3                && Variabele mVar2 met waarde 3 maken (in plaats van
                         && STORE- syntaxis)
? mVar1 * mVar2          && Dezelfde uitdrukking, met variabelen (resultaat is 15)
mVar1 = 6                && Variabele mVar1 bevat nu waarde 6
? mVar1 * mVar2          && Resultaat is 18
```

Evenals velden in een tabel hebben variabelen een gegevenstype. Het gegevenstype van een variabele is hetzelfde als het gegevenstype van de uitdrukking die u aan de variabele toewijst.

```
mVar1 = 5           && Numerieke uitdrukking 5 opslaan in variabele mVar1
? TYPE("mVar1")     && Resultaat is N
mVar2 = "Hallo"     && Tekenuitdrukking "Hallo" opslaan in variabele mVar2
? TYPE("mVar2")     && Resultaat is C
Z = DATE()          && Huidige datum opslaan in variabele Z
? TYPE("Z")         && Resultaat is D
```

Een variabele blijft in het geheugen totdat deze wordt *vrijgegeven*. U kunt een variabele expliciet vrijgeven met het commando RELEASE of CLEAR ALL. (Met CLEAR ALL worden alle variabelen vrijgegeven; met RELEASE worden alleen de opgegeven variabelen vrijgegeven.) Afhankelijk van het *bereik van variabelen worden variabelen op verschillende tijdstippen automatisch vrijgegeven*. In de sectie "Geheugenvariabelen definiëren", verderop in dit hoofdstuk, wordt het bereik van variabelen beschreven.

Naamgeving van geheugenvariabelen

In Help en in Hoofdstuk 1 van *Commando's en functies* wordt een overzicht gegeven van de regels voor de naamgeving van geheugenvariabelen. Naast deze regels gebruiken programmeurs doorgaans hun eigen conventies voor naamgeving. Als u conventies voor naamgeving gebruikt, vergroot u de leesbaarheid van uw programma's en kunt u code in een later stadium gemakkelijker wijzigen. Hierna volgen enkele conventies voor naamgeving die u mogelijk wilt gebruiken:

- Kies een naam die de inhoud of functie van de variabele aanduidt, zoals TotaalKosten of Teller.
- Gebruik als voorvoegsel van de naam van de variabele letters die het gegevenstype en het bereik van de variabele aangeven.

```
cNaam = "Severijn"       && Teken-variabele
nGewicht = 185           && Numerieke variabele
LOCAL lnTeller           && LOCAL-numerieke variabele
```

- Als de variabele correspondeert met een veld in een tabel, geeft u de variabele een soortgelijke naam, zoals mAdres voor een adresveld.

- Gebruik zowel hoofdletters als kleine letters. Bij namen van variabelen wordt geen onderscheid gemaakt tussen hoofdletters en kleine letters. U kunt derhalve hoofdletters en kleine letters door elkaar gebruiken. De hoofdletters in de volgende namen zorgen ervoor dat de namen gemakkelijker te lezen zijn:

```
nKlantRekTotaal = 129.54
cItemCode = "PK909"
```

Als eerste teken van een variabelenaam kunt u het onderstreepteken (_) gebruiken. In dBASE wordt dit teken echter gebruikt in de namen van systeemgeheugenvariabelen. U kunt dit teken daarom beter niet gebruiken in uw variabelen. Zie Hoofdstuk 24 voor meer informatie over systeemgeheugenvariabelen.

Geheugenvariabelen definiëren

In de meeste programmeertalen moet u geheugenvariabelen *definiëren* voordat u ze kunt gebruiken. Als u een variabele definieert, wijst u er een *bereik aan toe*. Dit bereik bepaalt de levensduur en beschikbaarheid van de variabele. In dBASE voor Windows kunt u variabelen maken zonder deze te definiëren, bijvoorbeeld met de opdracht `mvar=100`. Het is echter een goede gewoonte om variabelen te definiëren. (Array's moet u expliciet definiëren met het commando DECLARE. Zie "Werken met array's" verderop in dit hoofdstuk.)

dBASE voor Windows bevat vier commando's voor het definiëren van van het bereik van geheugenvariabelen: PUBLIC, PRIVATE, STATIC en LOCAL. Als u een variabele definieert met PUBLIC of STATIC, wordt de variabele automatisch gemaakt en wordt deze *geïnitialiseerd* met de waarde False (.F.). Als u een variabele definieert met PRIVATE of LOCAL, wordt de variabele niet automatisch gemaakt en geïnitialiseerd.

In de volgende voorbeelden wordt uitgelegd hoe variabelen worden gedefinieerd en geïnitialiseerd. Zie "Bereik van geheugenvariabelen" verderop in dit hoofdstuk voor gedetailleerde voorbeelden van het definiëren en gebruiken van variabelen.

```
PUBLIC mNaam              && mNaam definieren als PUBLIC. Variabele initialiseren als .F.
? mNaam                   && Resultaat is .F.
mNaam = SPACE(30)         && mNaam opgegeven waarde toewijzen
PRIVATE nKosten           && nKosten heeft nog geen waarde
? nKosten                 && Resultaat is foutmelding; nKosten heeft nog geen waarde
nKosten = 500             && nKosten wordt nu geinitialiseerd op 500
PUBLIC mCode, mBeschr     && Meerdere variabelen definiëren
LOCAL cNaam               && cNaam heeft nog geen waarde
STATIC lVoltooid          && Initialiseren als .F.
STATIC nTotaal = 1000     && Alleen variabelen van het type STATIC kunnen in één opdracht
                          && worden gedefinieerd en daarna geïnitialiseerd op een waarde
```

Bereik van geheugenvariabelen

Met het *bereik* van een geheugenvariabele wordt het volgende bepaald:

- De *levensduur van een variabele*, dat wil zeggen, wanneer de variabele wordt vrijgegeven.

- De *beschikbaarheid* van een variabele, dat wil zeggen, de omstandigheden waarin een subroutine de inhoud van de variabele kan benaderen of wijzigen.

Voor elk type variabele zijn er verschillende regels die de levensduur en de beschikbaarheid bepalen. Deze regels vindt u in Tabel 5.1. Zie het commando PUBLIC in *Commando's en functies* voor meer informatie over de overeenkomsten en verschillen tussen het bereik van geheugenvariabelen. Zie de voorbeeldprogramma's verderop in dit hoofdstuk voor voorbeelden van code waarin de regels voor het bereik van variabelen zijn toegepast.

Tabel 5.1 Levensduur en beschikbaarheid van geheugenvariabelen

Bereik	Levensduur	Beschikbaarheid
PUBLIC	Kan alleen expliciet worden vrijgegeven met RELEASE of CLEAR ALL	In alle subroutines, inclusief subroutines op hogere en lagere niveaus
PRIVATE	Wordt vrijgegeven wanneer de routine die de variabele heeft gemaakt, is uitgevoerd.	In de subroutine die de variabele heeft gemaakt en in subroutines op lagere niveaus
LOCAL	Wordt vrijgegeven wanneer de routine die de variabele heeft gemaakt, is uitgevoerd.	Alleen in de subroutine die de variabele heeft gemaakt
STATIC	Kan alleen expliciet worden vrijgegeven met RELEASE of CLEAR ALL	Alleen in de subroutine die de variabele heeft gemaakt

In vorige versies van dBASE werd het bereik PRIVATE gebruikt om de naam van een variabele te "verbergen" voor andere subroutines. In dBASE kunt u daarvoor beter het bereik LOCAL gebruiken.

Als u niet expliciet een bereik definieert, zijn variabelen die u in het commandovenster maakt automatisch PUBLIC en zijn variabelen die u in een programmabestand maakt automatisch PRIVATE. Voor eenvoudige programma's is het standaardbereik (PRIVATE) meestal voldoende. Bovendien kunt u de definitie overslaan zolang u elke variabele initialiseert in de subroutine waarin de variabele wordt gebruikt of in een subroutine op een hoger niveau. Als u gecompliceerde programma's schrijft, is het definiëren van het bereik van variabelen van groot belang voor de modulariteit en bruikbaarheid van uw programma's.

Als u in een programma naar een geheugenvariabele verwijst, zoekt dBASE eerst naar een beschikbare variabele van het type PRIVATE, LOCAL of STATIC met de opgegeven naam. Als deze niet wordt gevonden, zoekt dBASE naar een variabele van het type PUBLIC. In het volgende voorbeeld ziet u hoe dat in zijn werk gaat:

```
* Programma ABC
mVar = 100  && Variabele van type PRIVATE, LOCAL of STATIC met naam mVar zoeken
            && Als variabele gevonden, initialiseren op 100 (oude waarde overschrijven)
            && Als variabele niet gevonden, PUBLIC-variabele mVar zoeken
            && Als variabele gevonden, initialiseren op 100 (oude waarde overschrijven)
            && Als variabele mVar niet gevonden, PRIVATE-variabele mVar maken en
            && initialiseren op 100.

* Programma XYZ
mVar1 = mVar + 170    && Als hierboven: dBASE zoekt eerst naar een variabele van het type
                      && PRIVATE, LOCAL of STATIC met de naam mVar. Als die niet wordt
```

```
&& gevonden, zoekt dBASE naar een variabele van het type PUBLIC. Als
&& deze niet wordt gevonden, wordt een fout als resultaat gegeven.
```

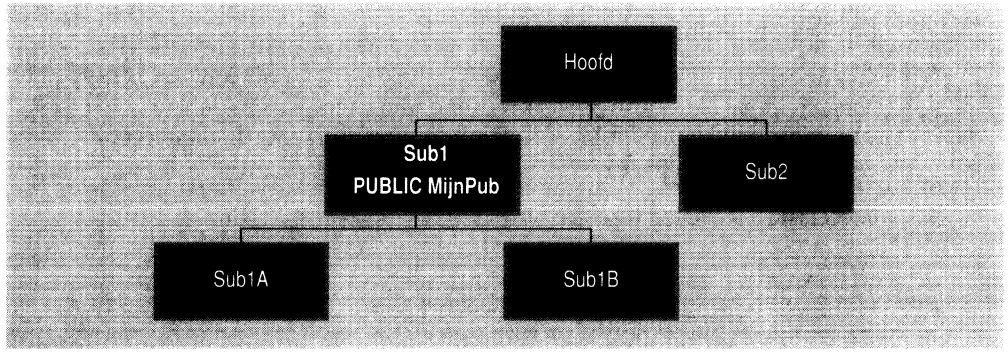
PUBLIC

Variabelen van het type PUBLIC, ook wel *globale* variabelen genoemd, hebben het grootste bereik. Deze variabelen zijn beschikbaar in alle subroutines en worden pas vrijgegeven als u RELEASE of CLEAR ALL gebruikt of dBASE afsluit.

Variabelen van het type PUBLIC zijn vooral geschikt voor instellingen die voor een hele applicatie gelden, zoals aanmeldingsnamen of standaardpaden voor gegevens. Deze variabelen zijn echter niet beschermd tegen overschrijven. Over het algemeen geldt dat in een goed ontworpen applicatie geen of zo weinig mogelijk variabelen van het type PUBLIC worden gebruikt.

In Afbeelding 5.1 wordt de beschikbaarheid van variabelen van het type PUBLIC toegelicht. `MijnPub`, die in Sub1 wordt gedefinieerd als een variabele van het type PUBLIC, is beschikbaar in alle subroutines. `MijnPub` wordt pas vrijgegeven wanneer u dat expliciet doet met RELEASE of CLEAR ALL.

Afbeelding 5.1 Beschikbaarheid van variabelen van het type PUBLIC

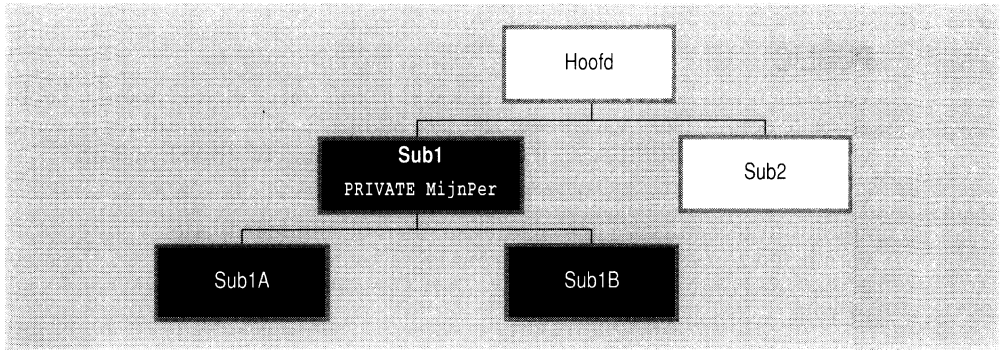


PRIVATE

Variabelen van het type PRIVATE zijn beschikbaar in de subroutine die de variabelen maakt en in alle subroutines op een lager niveau.

In Afbeelding 5.2 wordt de beschikbaarheid van variabelen van het type PRIVATE toegelicht. `MijnPers`, die in Sub1 wordt gedefinieerd als variabele van het type PRIVATE, is beschikbaar in Sub1, Sub1A en Sub1B. `MijnPer` wordt vrijgegeven nadat Sub1 is uitgevoerd.

Afbeelding 5.2 Beschikbaarheid van variabelen van het type PRIVATE



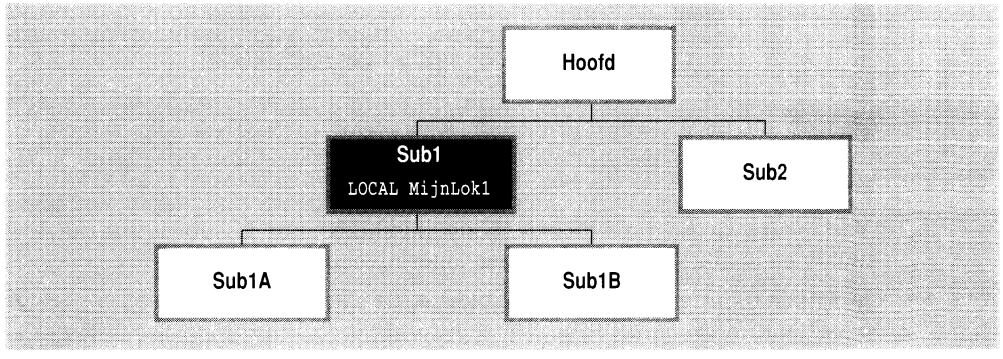
= Variabele is beschikbaar. De variabele wordt vrijgegeven nadat Sub1 is uitgevoerd.

LOCAL

Variabelen van het type LOCAL zijn alleen beschikbaar voor de procedure of functie waarin ze worden gedefinieerd. Omdat de beschikbaarheid van LOCAL-variabelen is beperkt tot één subroutine, wordt de definitie van LOCAL-variabelen ook wel *gegevens verbergen* genoemd

In Afbeelding 5.3 wordt de beschikbaarheid van variabelen van het type LOCAL toegelicht. `MijnLok1`, die in Sub1 wordt gedefinieerd als variabele van het type LOCAL, is alleen beschikbaar in Sub1. Telkens wanneer Sub1 wordt aangeroepen, wordt `MijnLok1` opnieuw geïnitialiseerd.

Afbeelding 5.3 Beschikbaarheid van variabelen van het type LOCAL



= Variabele is beschikbaar. De variabele wordt vrijgegeven nadat Sub1 is uitgevoerd.

Tijdelijke waarden in subroutines dienen over het algemeen te worden opgeslagen in variabelen van het type LOCAL. Voorbeelden hiervan zijn tellervariabelen in FOR...NEXT- of DO...WHILE-lussen.

STATIC

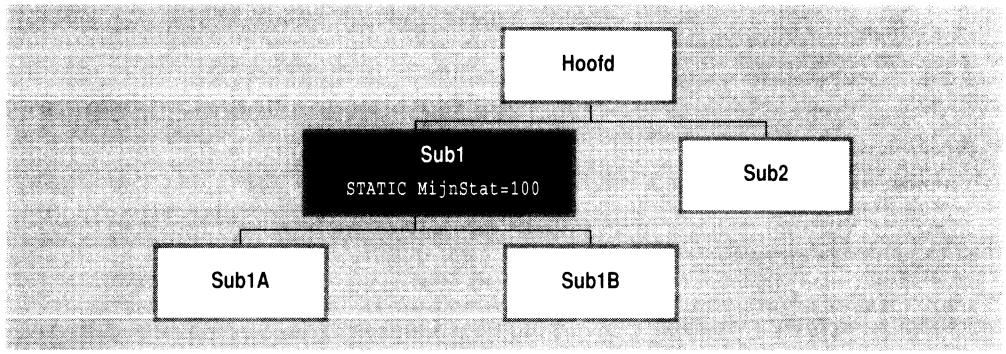
Variabelen van het type STATIC zijn lokaal wat beschikbaarheid betreft maar globaal wat levensduur betreft. Dat wil zeggen, een variabele van het type STATIC blijft bestaan als de subroutine waarin de variabele wordt gedefinieerd, is uitgevoerd (net als bij variabelen van het type PUBLIC). De variabele is echter pas weer beschikbaar wanneer u de subroutine opnieuw aanroept. Evenals een variabele van het type LOCAL, is een variabele van het type STATIC alleen beschikbaar in de subroutine die de variabele heeft gemaakt. In tegenstelling tot een variabele van het type LOCAL, krijgt een variabele van het type STATIC echter de oude waarde terug wanneer de subroutine wordt aangeroepen die de variabele heeft gemaakt. Variabelen van het type STATIC zijn derhalve het meest geschikt voor het opslaan van waarden die steeds toenemen, zoals subtotalen in rapporten.

In tegenstelling tot andere typen variabelen kunt u variabelen van het type STATIC in één opdracht definiëren *en initialiseren*. Daartoe gebruikt u de volgende syntaxis:

```
STATIC <geheugenvar> = <waarde>
```

In Afbeelding 5.4 wordt de beschikbaarheid van variabelen van het type STATIC toegelicht. `MijnStat`, die in Sub1 wordt gedefinieerd als een variabele van het type STATIC, is alleen beschikbaar in Sub1. De eerste keer dat Sub1 wordt aangeroepen, wordt `MijnStat` geïnitialiseerd op de waarde 100. De volgende keren dat Sub1 wordt aangeroepen, wordt `MijnStat` niet opnieuw geïnitialiseerd. De waarde van `MijnStat` wordt zodoende telkens opnieuw gebruikt.

Afbeelding 5.4 Beschikbaarheid van variabelen van het type STATIC



= Variabele is beschikbaar en blijft beschikbaar nadat Sub1 en Hoofd zijn uitgevoerd

In het volgende voorbeeld ziet u hoe u de toegang tot geheugenvariabelen kunt beperken of juist vrijmaken door de geheugenvariabelen te definiëren als variabelen van het type PUBLIC, PRIVATE of LOCAL. Zoals u in de code kunt zien, kunnen variabelen van het type PRIVATE niet worden gedefinieerd als variabelen van het type PUBLIC in een subroutine op een lager niveau. Variabelen van het type LOCAL kunnen wel als variabelen van het type PUBLIC worden gedefinieerd in een subroutine op een lager niveau. Wijzigingen die u aanbrengt in variabelen die u als PUBLIC definieert in een subroutine, worden niet doorgegeven aan een hoger programmaniveau waar dezelfde variabelen zijn gedefinieerd als LOCAL.

```
* Programma Prog1
PUBLIC c_ID             && Beschikbaar in alle subroutines
c_ID = "Mijn naam"
nState = "OR"           && PRIVATE-variabele (standaardinstelling)
LOCAL dDatum            && Niet beschikbaar in subroutines op lagere niveaus
dDatum = {12-12-95}
Do prog2
? c_ID                  && Resultaat is "Mijn naam" (variabelen met dezelfde naam in Prog2
                        && zijn PRIVATE)
? nTellen               && Resultaat is 45 (nTellen is PUBLIC in Prog2 en heeft geen
                        && definitie in Prog1w
? dDatum                && Resultaat is {12/12/95} omdat dDatum is LOCAL in Prog1 en
                        && LOCAL-variabelen niet kunnen worden gewijzigd in subroutines op
                        && lager niveau
* Programma Prog2
PUBLIC nTellen, dDatum && nTellen is nog niet gedefinieerd, dDatum is LOCAL in Prog1
PUBLIC nSTATE           && Resultaat is foutwaarde omdat nState is geïnitialiseerd als
                        && PRIVATE in Prog1
PRIVATE c_ID            && Nieuwe variabele die niet strijdig is met de bestaande variabele
                        && van het type PUBLIC met dezelfde naam
c_ID = "Naam van deze persoon"
nTellen = 45
dDatum = {04-03-95}
RETURN
```

Belangrijk Als u variabelen van het type STATIC in één opdracht definieert en initialiseert, worden variabelen van het type STATIC in subroutines alleen de eerste keer geïnitialiseerd wanneer de subroutine wordt aangeroepen. De volgende keren dat de subroutine wordt aangeroepen, worden de variabelen van het type STATIC niet opnieuw geïnitialiseerd. In het volgende voorbeeld wordt dit toegelicht.

```
* Programma Prog1
PUBLIC nTotaal
nTotaal = 1000
Do Prog2
⋮
Do Prog2

* Programma Prog2
STATIC nVerkopen = 500   && nVerkopen initialiseren op 500 wanneer Prog2 de eerste keer
                         && wordt aangeroepen
STATIC nWinPct
nWinPct = .03            && nWinPct initialiseren op 0,03 wanneer Prog2 wordt aangeroepen
? nVerkopen              && Resultaat is 500 wanneer Prog2 voor de eerste keer wordt
                         && aangeroepen
                         && Resultaat is 800 (500 + 300) wanneer Prog2 voor de tweede keer
                         && wordt aangeroepen, omdat nVerkopen verderop in dit programma
                         && met 300 wordt verhoogd
STATIC nMarge            && Wordt alleen geïnitialiseerd op .F. wanneer Prog2 voor de
                         && eerste keer wordt uitgevoerd
IF TYPE("nMarge") = "L"  && Resultaat is waar wanneer nMarge voor de eerste keer wordt
                         && geïnitialiseerd
   nMarge = 0            && nMarge wijzigen in numerieke variabele
```

```
ENDIF
nMarge = nMarge + (nVerkopen * nWinPct)
? nMarge                    && Resultaat is 15 (0+(500*0,03)) wanneer Prog2 voor de eerste
                            && keer wordt uitgevoerd
                            && Resultaat is 39 (15+(800*0,03)) wanneer Prog2 voor de tweede
                            && keer wordt uitgevoerd
nTotaal = nTotaal + nVerkopen
? nTotaal                   && Resultaat is 1500 (1000 + 500) wanneer Prog2 voor de eerste
                            && keer wordt aangeroepen
                            && Resultaat is 2300 (1500 + 800) wanneer Prog2 voor de tweede
                            && keer wordt aangeroepen
nVerkopen = nVerkopen + 300
RETURN
```

Werken met array's

Een *array* is een groep gerelateerde geheugenvariabelen die zijn ingedeeld in rijen en kolommen. De afzonderlijke variabelen worden de *elementen* of *leden* van de array genoemd. De interne structuur van een array is op ongeveer dezelfde wijze ingedeeld als die van een spreadsheet, waarbij elk array-element equivalent is aan een cel in de spreadsheet. Evenals andere geheugenvariabelen kunt u array's als parameters doorgeven aan functies, waarna waarden als resultaat worden gegeven.

Programmeurs gebruiken array's doorgaans voor de volgende activiteiten:

- Een reeks gerelateerde geheugenvariabelen verwerken in meerdere doorgangen
- Werken met waarden in een tabel zonder de tabel geopend te houden, wat bijzonder handig is in een multi-user-omgeving
- Tijdelijke "tabellen" met informatie opslaan met een indeling in rijen en kolommen

Array's kunnen een, twee of meer dimensies hebben. 1-dimensionale array's bevatten één rij en een opgegeven aantal kolommen. 2-dimensionale array's bevatten een opgegeven aantal rijen en kolommen. dBASE voor Windows ondersteunt array's met meer dan twee dimensies. De meeste array-functies kunnen echter alleen worden uitgevoerd op 1- of 2-dimensionale array's. 1- en 2-dimensionale array's zijn de typen die meestal door programmeurs worden gebruikt.

U gebruikt het commando DECLARE als u een array wilt maken of initialiseren. Als u een 1-dimensionale array met zes elementen wilt initialiseren, gebruikt u het volgende commando:

```
DECLARE aArrayNaam[6]       && Vierkante haken [ ] zijn vereist bij het werken met array's
```

Als u een 2-dimensionale array met drie rijen en vier kolommen voor in totaal twaalf elementen wilt initialiseren, gebruikt u het volgende commando:

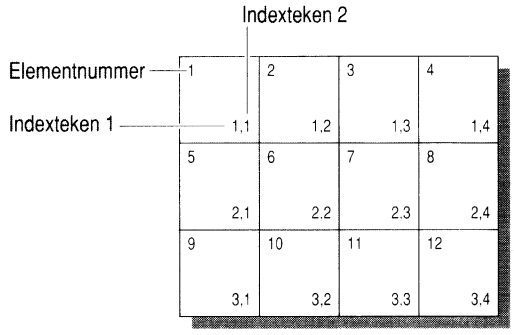
```
DECLARE aArrayNaam[3,4]
```

Wanneer een array is geïnitialiseerd, hebben alle elementen in de array de waarde .F. (False). U kunt echter waarden met een willekeurig gegevenstype in een element in de array plaatsen. Dat wil zeggen, één array kan datumgegevens, numerieke gegevens,

reeksgegevens en alle andere typen gegevens bevatten die zijn toegestaan in een geheugenvariabele.

U kunt op twee manieren verwijzen naar afzonderlijke elementen in een array: u kunt de *indextekens* of het *nummer* van het element gebruiken. De *indextekens* van een element geven de rij en kolom aan waarin het element zich bevindt. Het *nummer* van een element geeft de sequentiële positie aan van de elementen in de array. Array-elementen worden opeenvolgend genummerd vanaf de eerste rij en de eerste kolom van de array. In Afbeelding 5.5 ziet u het elementnummer en de indextekens voor elk element in een array dat drie rijen en vier kolommen bevat.

Afbeelding 5.5 Elementnummers en indextekens van elementen in een 2-dimensionale array



U verwijst naar een afzonderlijk element in een array door de indextekens van het element te gebruiken:

```
aArray[3,3] = 50      && Sla 50 op in de derde rij, derde kolom
```

Array's en waarden in array's manipuleren

dBASE bevat vele functies voor het manipuleren van array's en waarden in array's. Met deze functies kunt u array's gebruiken als een dynamische gegevensstructuur die lijkt op tabellen. Bovendien kunt u met array's werken als klassen van objecten. (Zie "Werken met array's als objecten" in Hoofdstuk 10.)

In Tabel 5.2 wordt een overzicht gegeven van de dBASE-commando's en functies voor array's.

Tabel 5.2 Overzicht van array-taken en bijbehorende commando's en functies

Array-taak	Functies	Beschrijving
Verwijzen naar elementen	ASUBSCRIPT()	Geeft het indexteken van de rij of kolom van een opgegeven element als resultaat, gegeven het elementnummer.
	AELEMENT()	Geeft het elementnummer van een opgegeven element als resultaat, gegeven de rij- en kolom-indextekens van het element.
Opvullen met waarden	AFILL()	Vult een, sommige of alle elementen in een array op met een opgegeven waarde.

Tabel 5.2 Overzicht van array-taken en bijbehorende commando's en functies (vervolg)

Array-taak	Functies	Beschrijving
	ADIR()	Vult een array op met de lijst met directory's van een opgegeven bestandsschema.
	AFIELDS()	Vult een array op met de structuurinformatie over een geopende tabel.
Zoeken naar waarden	ASCAN()	Zoekt in een array naar de opgegeven uitdrukking.
Elementen toevoegen en verwijderen	ADEL()	Verwijdert een element, of een rij of kolom elementen, zonder de dimensies van de array te wijzigen.
	AGROW()	Voegt een element, of een rij of kolom elementen, toe en wijzigt de dimensies van de array. Met AGROW() kunt u van een 1-dimensionale array een 2-dimensionale maken.
	AINS()	Voegt een element, of een rij of kolom elementen, in zonder de dimensies van de array te wijzigen.
	ARESIZE()	Vergroot of verkleint de array. Met ARESIZE() kunt u van een 1-dimensionale array een 2-dimensionale maken.
Elementen kopiëren en sorteren	ACOPY()	Kopieert elementen van het ene array naar het andere.
	ASORT()	Sorteert de elementen in een 1-dimensionale array of de rijen in een 2-dimensionale array.
De grootte bepalen	ALEN()	Geeft het aantal elementen, rijen of kolommen in een opgegeven array als resultaat.
Kopiëren tussen array's en tabellen	COPY TO ARRAY	Kopieert gegevens van een databasetabel naar een array
	APPEND FROM ARRAY	Voegt gegevens uit een array toe aan een databasetabel
	REPLACE FROM ARRAY	Vervangt de gegevens in een databasetabel door gegevens in een array.

In de volgende voorbeelden ziet u hoe u enkele van de array-functies uit Tabel 5.2 kunt gebruiken.

AFILL(), ASORT()

In het volgende voorbeeld ziet u hoe u tabelwaarden kunt opslaan in een array in een multi-user-omgeving. Deze techniek is handig wanneer u tabelwaarden wilt lezen zonder daarbij te worden gehinderd door record- of bestandsvergrendelingen die andere gebruikers mogelijk voor de tabel hebben ingesteld. De waarden die met behulp van deze techniek als resultaat worden gegeven, geven echter niet de wijzigingen weer die in de tabel zijn aangebracht nadat u deze hebt gesloten. Gebruik deze techniek daarom alleen wanneer de waarden die als resultaat worden gegeven, niet de meest recente gegevens in de tabel hoeven weer te geven.

```
USE <Bestandsnaam> EXCLUSIVE        && EXCLUSIVE zorgt voor toegang tot alle records in
                                    && een multi-user-omgeving

COUNT TO nRecords
DECLARE aArray[nRecords,3]          && 1 rij voor elke record in tabel en 3 kolommen maken
AFILL(aArray,0)                     && Plaats een 0 in alle array-elementen
GO TOP
nRec = 1
```

```
SCAN
   aArray[nRec,1] = Salaris          && Salaris in eerste kolom
   aArray[nRec,2] = Dienstjaren      && Dienstjaren in tweede kolom
   aArray[nRec,3] = Leeftijd         && Leeftijd in derde kolom
   nRec = nRec + 1
ENDSCAN
USE                          && Andere gebruikers toegang tot de tabel geven
ASORT(aArray)                && ASORT() vereist dat alle elementen die worden gesorteerd,
                             && hetzelfde gegevenstype hebben. Daarom is eerder AFILL gebruikt.
? aArray[nRecords,1]         && Hoogste salaris is in de eerste kolom van de laatste rij
ASORT(aArray,3)              && Sorteren op leeftijd, de waarde in de derde kolom
? aArray[1,3]                && Resultaat is de jongste werknemer
? aArray[nRecords,3]         && Resultaat is de oudste werknemer
```

ARESIZE()

In dit voorbeeld ziet u hoe u de grootte van de array in het eerste voorbeeld kunt wijzigen, zodat u meer waarden kunt toevoegen. U wilt twee kolommen aan de array toevoegen:

```
ARESIZE(aArray,nRecords,5,1)  && Array heeft nu 5 kolommen
                              && 1 geeft aan dat de positie van de oorspronkelijke
                              && waarden niet mag worden gewijzigd
```

ALEN(), AGROW()

In dit voorbeeld wilt u te weten komen hoeveel rijen of kolommen de array bevat, zodat u bepaalde functies kunt verrichten. U weet dat het salaris van iedere werknemer in de eerste kolom staat, de voornaam in de vijfde kolom, de achternaam in de zesde kolom en de geboortedatum in de laatste kolom. U wilt alle werknemers een salarisverhoging geven en bovendien wilt u verjaardagskaarten sturen naar iedereen die deze maand jarig is. U maakt een tweede array voor het opslaan van de namen van mensen die u een kaart wilt sturen.

```
DECLARE aVerjaardag[1,2]  && Array maken voor opslag van namen mensen die kaart krijgen
nNieuweRij = 1

nRijen = ALEN(aArray,1)                          && Resultaat is aantal rijen in werknemer-
                                                 && array
nKolmn = ALEN(aArray,2)                          && Resultaat is aantal kolommen in
                                                 && werknemer-array
nHuiMaand = MONTH(DATE())                        && Resultaat is huidige maand (1-12)

FOR mVar = 1 TO nRijen                           && Elke rij in werknemer-array doorlopen
   aArray[mVar,1] = aArray[mVar,1] * 1.03        && Iedereen 3% opslag
   IF MONTH(aArray[mVar,nCols]) = nHuiMaand      && Geboortemaand met huidige maand
                                                 && vergelijken
      aVerjaardag[nNieuweRij,1] = aArray[mVar,5]&& Voornaam in eerste kolom van kaart-
                                                 && array
      aVerjaardag[nNieuweRij,2] = aArray[mVar,6]&& Achternaam in tweede kolom van
                                                 && kaart-array
      AGROW(aVerjaardag,1)                       && 1 rij toevoegen aan kaart-array
      nNieuweRij = nNieuweRij + 1
   ENDIF
NEXT
```

Tekenvariabelen uitbreiden

Soms weet u pas welke argumenten u op een opdrachtregel moet gebruiken als uw programma al actief is. U schrijft bijvoorbeeld een subroutine waarmee woorden in een tabel worden verwerkt op basis van criteria die de gebruiker invoert, zoals `Prov = "NH"`. In dit geval kunt u geen criteria in de broncode invoeren omdat u niet weet wat de criteria zullen zijn. Bovendien moet u bij sommige commando's, zoals FIND, letterlijke waarden gebruiken als argumenten, terwijl u meestal pas tijdens runtime weet welke waarde u moet invoeren. Voor deze situaties beschikt dBASE over de *macro-operator* (&).

De macro-operator vervangt de naam van een variabele op de commandoregel door de inhoud van de teken-geheugenvariabele. Op deze manier kunt u commandoregel-argumenten opslaan in tekenvariabelen en kunt u de argumenten met behulp van de macro-operator afleiden tijdens runtime. Dit proces wordt *macrovervanging* of *macro-uitbreiding* genoemd.

U voert een macrovervanging uit door de macro-operator (&) direct voor de naam van een variabele op een commanodregel te gebruiken. Tijdens runtime wordt de inhoud van de tekenvariabele geëvalueerd en worden het &-teken en de naam van de variabele vervangen door het resultaat van de evaluatie, zoals u in het volgende voorbeeld kunt zien.

```
lVoorwaarde = "verkopen > 10000"
COUNT FOR &lVoorwaarde
```

De syntaxis voor het gebruik van de macro-operator is als volgt:

```
&<teken-geheugenvariabele>[.]
```

De optionele punt (.) achter de naam van de variabele geeft het einde van de naam van de variabele aan. Deze punt wordt ook wel *macro-afsluiting* genoemd. Gebruik de punt als u de naam van de variabele wilt onderscheiden van tekst die erop volgt op de opdrachtregel.

In dBASE-programma's gelden de volgende beperkingen ten aanzien van het gebruik van macro-uitbreidingen:

- Aangezien macrovervanging tijdens runtime plaatsvindt, kunnen de macro's niet worden uitgebreid met preprocessor-instructies, die tijdens het compileren worden geëvalueerd. Zie Hoofdstuk 7 voor meer informatie over preprocessor-instructies.
- U kunt de macro's alleen gebruiken voor namen van variabelen, niet voor veldnamen.
- Met behulp van macrovervanging kunt u commando's invoeren. Dit wordt echter niet aanbevolen, aangezien hierdoor problemen kunnen ontstaan met gecompileerde versies van applicaties en de programma-uitvoering wordt vertraagd.
- Macro-uitbreiding kan niet worden toegepast op variabelen van het type LOCAL of STATIC. Dit wordt in het volgende voorbeeld toegelicht.

```
LOCAL mVar1
mVar1 = "waarde > 1000"
SET FILTER TO &mvar1     && Geeft een syntaxisfout als resultaat
```

Hier volgen enkele voorbeelden van macro-uitbreiding:

In het volgende voorbeeld moet u een SET-waarde wijzigen en wilt u na afloop de oorspronkelijke waarde weer herstellen. Hiervoor worden vaak macro's gebruikt.

```
cVeilig = SET("SAFETY")          && Geeft "ON" of "OFF" als resultaat
SET SAFETY OFF
** typ hier uw commando's
SET SAFETY &cVeilig              && Omgeving instellen op beginwaarden
```

In het volgende voorbeeld wordt een dynamisch filter gemaakt op basis van de voorkeuren van de gebruiker:

```
cCriteria = GETEXPR("","Stel criteria op", "L")   && Gebruiker kan logische voorwaarde
                                                  && maken
IF LEN(cCriteria) <> 0                            && Gebruiker heeft bewerking niet
                                                  && geannuleerd of niets opgegeven voor
                                                  && uitdrukking
   SET FILTER TO &cCriteria                       && Tabel filteren volgens criteria van
                                                  && gebruiker
ENDIF
```

In het volgende voorbeeld heeft de gebruiker in een keuzelijst een rapport geselecteerd dat moet worden gestart. Vervolgens heeft de gebruiker het keuzerondje "Printer" geselecteerd. U hebt de keuzen van de gebruiker opgeslagen in de variabelen `cRapport` en `cWaar`.

```
cRapport = <naam van het rapport dat de gebruiker heeft geselecteerd>
cWaar = "TO PRINT"
REPORT FORM &cRapport &cWaar
```

Geheugenvariabelen opslaan in bestanden

U kunt geheugenvariabelen, inclusief array's, opslaan in bestanden en deze later opnieuw laden. Programmeurs gebruiken geheugenbestanden (met de extensie .MEM) doorgaans voor de opslag van gegevens, zoals programmaconfiguratie of printerinstellingen, die in één record in een tabel passen. Met de commando's SAVE en RESTORE maakt u geheugenbestanden en haalt u deze op.

- Met SAVE slaat u enkele of alle geheugenvariabelen op in een geheugenbestand. Gebruik een naamschema in combinatie met jokertekens (?, *) als u alleen bepaalde variabelen wilt opslaan.

Met SAVE kunt u geen variabelen van de volgende typen opslaan: functie-aanwijzer, objectverwijzing of systeemgeheugenvariabelen.

- Met RESTORE laadt u de variabelen vanuit een geheugenbestand weer in het geheugen. RESTORE geeft alle bestaande geheugenvariabelen eerst vrij voordat de variabelen in het geheugenbestand opnieuw worden geladen (standaardinstelling). Als u bestaande variabelen wilt behouden, gebruikt u de optie ADDITIVE.

In het volgende voorbeeld ziet u een eenvoudig gebruik van de commando's SAVE en RESTORE:

```
* Hoofdprogramma van demoversie van software
IF FILE("Opgslvar.mem")         && Bestaat het geheugenbestand?
   RESTORE FROM Opgslvar        && Zo ja, gebruik het dan.
ELSE                            && Zo nee,
   dEersteStart = DATE()        && sla dan de datum op waarop het programma voor het eerst
                                && gestart
   SAVE TO Opgslvar
ENDIF
IF DATE() - dEersteStart >= 30 && Is het meer dan 30 dagen geleden dat het programma voor
                                && het eerst is gestart?
   ** geef een formulier weer voor registratie na 30 dagen gebruik
ENDIF
```

Alle waarden die RESTORE als resultaat geeft, zijn van het type PRIVATE (standaardinstelling). Als u wilt dat deze waarden een ander bereik hebben, moet u het bereik definiëren voordat u het commando RESTORE gebruikt. Bovendien moet u in dat geval de optie ADDITIVE gebruiken. Dit kunt u zien in het volgende voorbeeld, waarin het bestand met geheugenvariabelen wordt gebruikt dat in het vorige voorbeeld is opgeslagen.

```
* Programma 1 -- dit werkt niet
PUBLIC dEersteStart
IF FILE("Opgslvar.mem")
   RESTORE FROM opgslvar        && dEersteStart is nu van het type PRIVATE, ondanks het
                                && eerdere commando PUBLIC
ENDIF
RETURN

* Programma 2 -- dit werkt wel
PUBLIC dEersteStart
IF FILE("Opgslvar.mem")
   RESTORE FROM opgslvar ADDITIVE    && dEersteStart is nu van het type PUBLIC
ENDIF
RETURN
```

Hoofdstuk

6

Werken met gegevenstypen

dBASE ondersteunt verschillende gegevenstypen voor geheugenvariabelen en velden en bevat functies voor het manipuleren en opmaken van gegevens van elk type. In dit hoofdstuk worden gegevenstypen geïntroduceerd en worden de meest voorkomende gegevenstypen besproken. Dat wil zeggen, gegevenstypen die zowel voor variabelen als velden kunnen worden gebruikt. Gegevenstypen die alleen van toepassing zijn op variabelen, worden beschreven in Hoofdstuk 5. Gegevenstypen die alleen van toepassing zijn op velden, worden beschreven in Hoofdstuk 18.

Zie *Commando's en functies* voor meer informatie over de commando's en functies die in dit hoofdstuk worden gebruikt. In Hoofdstuk 1 van *Commando's en functies* worden de regels beschreven die gelden voor de verschillende gegevenstypen.

Over gegevenstypen

Een *gegevenstype* is een classificatie voor gegevens die zijn opgeslagen in een veld of geheugenvariabele. Voor elk gegevenstype gelden regels die bepalen hoe u met de gegevens werkt en welk gegevenstype u kunt gebruiken in het veld of de variabele. Gegevens van het type Numeriek kunnen bijvoorbeeld alleen getallen zijn, waarmee u berekeningen kunt uitvoeren. Gegevens van het type Teken kunnen letters, getallen of leestekens zijn. U kunt echter geen berekeningen uitvoeren op deze getallen.

In vorige versies van dBASE zijn gegevenstypen alleen van toepassing op de gegevens waarmee de gebruiker werkt, zoals namen, hoeveelheden of datums. In dBASE voor Windows zijn gegevenstypen ook van toepassing op bepaalde gegevens waarmee programmeurs werken, zoals objecten, procedures of codeblokken. Een variabele die verwijst naar een object is bijvoorbeeld van het type Object. Een variabele die verwijst naar een procedure of functie is van het type Functie-aanwijzer. Een variabele die een uitdrukking of reeks commando's bevat, is van het type Codeblok. Zie Hoofdstuk 4 voor meer informatie over het werken met codeblokken en functie-aanwijzers.

Evenals voor de traditionele gegevenstypen gelden voor deze nieuwe gegevenstypen regels die bepalen hoe u met deze gegevenstypen werkt. U kunt een codeblok toewijzen aan een kenmerk en u kunt een codeblok doorgeven als een parameter.

Met deze nieuwe gegevenstypen is het concept verruimd van wat in applicaties als gegevens wordt beschouwd. In vorige versies van dBASE zijn gegevens de elementen waarmee de gebruiker in tabellen werkt. In dBASE voor Windows zijn gegevens alle informatie waarmee een handeling kan worden uitgevoerd. Deze informatie kan iemands adres zijn dat u afdrukt of een knop waarop u klikt.

Tabel 6.1 bevat een overzicht van de gegevenstypen die in dBASE worden ondersteund.

Tabel 6.1 Gegevenstypen in dBASE

	Gegevenstype	Symbool
Variabelen, of velden in dBASE-tabellen	Teken	C
	Datum	D
	Logisch (Boole)	L
	Numeriek	N
Alleen variabelen	Bladwijzer	BM
	Codeblok	CB
	Functie-aanwijzer	FP
	Object	O
	Array	A
	Binair	B
Alleen velden in dBASE-tabellen	Zwevend (zwevend decimaalteken)	F
	Memo	M
	OLE	O

Standaardinstellingen voor gegevensopmaak

In dBASE zijn geen standaardinstellingen voor de opmaak van datums, tijden, valuta en getallen gedefinieerd. In plaats daarvan worden de standaardinstellingen gebruikt die u opgeeft met de optie Internationaal van Configuratiescherm. Configuratiescherm is een Windows-hulpmiddel voor het aanpassen en instellen van de Windows-omgeving. Bij de meeste Windows-toepassingen, met name de toepassingen die in meerdere landen worden gebruikt, kan de gebruiker de opmaak zelf instellen in Configuratiescherm en hoeft dus geen keuze te maken uit vooraf gedefinieerde standaardinstellingen.

Als u nog nooit met Configuratiescherm hebt gewerkt, zoekt u eerst het desbetreffende pictogram in de Hoofdgroep. Raadpleeg de Windows-documentatie voor meer informatie over het gebruik van Configuratiescherm.

U kunt de standaardinstellingen in Configuratiescherm op drie manieren wijzigen:

- Gebruik het desbetreffende commando SET, zoals SET CENTURY of SET SEPARATOR. Instellingen die u op deze manier wijzigt, zijn tijdelijk. Als u dBASE afsluit, worden de standaardinstellingen hersteld.
- Wijzig de instellingen in DBASEWIN.INI, het configuratiebestand van dBASE. Hiertoe gebruikt u SET of kiest u **Kenmerken | Bureaublad** als u de instellingen interactief wilt wijzigen. U kunt DBASEWIN.INI ook rechtstreeks bewerken. Instellingen die u op deze manier wijzigt, zijn permanent (totdat u de instellingen opnieuw wijzigt). Telkens wanneer u dBASE start, worden de instellingen in Configuratiescherm vervangen door deze instellingen. Zie Appendix C in het *Handboek* voor volledige informatie over de instellingen in DBASEWIN.INI.
- Start dBASE met de optie -c als u een ander configuratiebestand dan DBASEWIN.INI wilt opgeven. Dit wordt eveneens beschreven in Appendix C in het *Handboek*.

Werken met gegevens van het type Teken

Het gegevenstype Teken is het gegevenstype dat het meest in programma's wordt verwerkt. Gegevens van het type Teken bestaan uit letters, getallen, leestekens en andere symbolen, inclusief spaties. In dBASE worden gegevens van het type Teken beschouwd als reeksen van symbolen. Dientengevolge worden tekenvelden of geheugenvariabelen vaak "reeksen" of "reeksgegevens" genoemd.

Hoewel tekenreeksen getallen kunnen bevatten, kunt u met deze getallen geen wiskundige berekeningen uitvoeren. Evenals letters en leestekens worden getallen in dBASE beschouwd als symbolen. In tabellen worden postcodes en telefoonnummers doorgaans opgeslagen in tekenvelden, aangezien u hiermee geen berekeningen hoeft uit te voeren. Omgekeerd worden bedragen en hoeveelheden die u wilt optellen of waarvan u het gemiddelde wilt berekenen, opgeslagen in numerieke of zwevende velden.

U moet scheidingstekens gebruiken voor reeksen tekens in uitdrukkingen. U kunt enkele aanhalingstekens ('), dubbele aanhalingstekens (") of vierkante haken ([ ]) gebruiken als scheidingstekens. U moet hetzelfde scheidingsteken gebruiken aan beide zijden van de reeks. Voorbeeld: `[verkopen]` is een geldige tekenuitdrukking, maar `"verkopen]` niet.

```
Naam = "Spijker"      && Reeks "Spijker" opslaan in tekenvariabele Naam
Klantnr = '1234'      && Reeks "1234" opslaan in tekenvariabele Klantnr
Hoeveelh= 1234        && Waarde 1234 opslaan in numerieke variabele Hoeveelh
```

In Tabel 6.2 worden enkele voorbeelden gegeven van de manier waarop u gegevens van het type Teken kunt manipuleren in programma's.

Tabel 6.2 Tekenreeksen manipuleren

Taak	Functie	Voorbeeld
De lengte opmeten	LEN()	LEN(Ond_naam)
De positie bepalen van een subreeks in een grotere reeks	AT() RAT()	AT("Din","Maandag, Dinsdag,...")
Een subreeks ophalen	SUBSTR()	SUBSTR(Ond_nr,7,4)

Tabel 6.2 Tekenreeksen manipuleren (vervolg)

Taak	Functie	Voorbeeld
Ophalen vanaf linkerkant	LEFT()	LEFT(Ond_naam,9)
Ophalen vanaf rechterkant	RIGHT()	RIGHT(mOnd_nr,4)
Een reeks in een andere invoegen	STUFF()	STUFF(mVoorwaarden,8,2,mDagen)
Tekens herhalen	REPLICATE() SPACE()	REPLICATE("*",16) mNaam = SPACE(30)
Reeksen aaneenschakelen	+ operator – operator	mVoornaam + mAchternaam, of mVoornaam–mAchternaam
Reeksen vergelijken met behulp van jokertekens	LIKE()	LIST Ond_nr FOR LIKE("*-700-??30",Ond_nr)

Tekenreeksen vergelijken

Een veelvoorkomende taak bij het werken met gegevens van het type Teken is het vergelijken van de inhoud van twee reeksen. Meestal voert u reeksvergelijkingen uit als u naar gegevens zoekt of als u spronginstructies wilt maken voor de programma-uitvoering. Gebruik relationele operatoren (zoals >, <, =) als u reeksen wilt vergelijken. In Hoofdstuk 1 van *Commando's en functies* worden de relationele operatoren beschreven die worden gebruikt voor reeksvergelijkingen.

In applicaties is het vaak noodzakelijk om te weten of een gebruiker iets heeft getypt in een invoervak. Stel dat uw applicatie een dialoogvenster weergeeft waarin een naam moet worden getypt. Het dialoogvenster bevat een invoervak Naam en een knop OK. Als de gebruiker het dialoogvenster sluit, moet de applicatie controleren of de gebruiker een naam heeft ingevoerd. Als dat het geval is, moet de naam worden opgezocht in een tabel.

```
IF ISBLANK(Form.Naam.Value)
   RETURN
ELSE
   SCAN FOR Name = Form.Naam.Value
      ? Naam
   ENDSCAN
ENDIF
```

Fonetisch vergelijken

Reeksen teken voor teken vergelijken werkt goed als beide reeksen de juiste spelling hebben. Spel- en typefouten zijn vaak echter niet te voorkomen. Als u overeenkomsten wilt zoeken tussen twee soortgelijke reeksen die spelfouten bevatten, vergelijkt u de reeksen op basis van klank. Als u reeksen fonetisch wilt vergelijken, genereert u een *soundex*-code (code voor klankovereenkomst) voor elke reeks. Soundex is een veel gebruikte formule voor het genereren van de fonetische waarde van woorden. dBASE bevat twee functies voor het genereren en vergelijken van soundex-waarden:

- SOUNDEX() geeft een soundex-code voor een tekenreeks als resultaat
- DIFFERENCE() geeft een getal (0–4) als resultaat dat de mate van fonetische overeenkomst tussen twee reeksen aangeeft

Soundex-codes zijn taalspecifiek. Dat wil zeggen, elk taalaansturingsprogramma bevat een eigen formule voor het genereren van soundex-codes. Zie Appendix C voor meer informatie over taalaansturingsprogramma's.

Dit zijn enkele voorbeelden van soundex-vergelijkingen:

```
? "Jansen"="Janssens"                    && Geeft .F. als resultaat
? SOUNDEX("Jansen")=SOUNDEX("Janssens")  && Geeft .T. als resultaat
? DIFFERENCE("Jansen","Janssens")        && Geeft 4 als resultaat (veel overeenkomst)
? DIFFERENCE("Moniek","Mona")            && Geeft 3 als resultaat (minder overeenkomst)
? DIFFERENCE("dBASE","C")                && Geeft 1 als resultaat (nauwelijks
                                         && overeenkomst)
```

Met de functie SOUNDEX() kunt u een index maken op basis van soundex-waarden. In de volgende voorbeelden worden SOUNDEX() en DIFFERENCE() gebruikt om te zoeken naar gelijkluidende bedrijfsnamen als het commando INDEX geen exacte overeenkomst oplevert. Als minstens één gelijkluidende naam wordt gevonden, wordt het resultaat weergegeven. De bedrijfsnaam bevindt zich in het invoervak BedrNaam.

```
USE Accounts
INDEX ON SOUNDEX(Bedrijf) TAG SoundComp          && SOUNDEX()-index maken
INDEX ON Bedrijf TAG Bedrijf
IF SEEK(Form.BedrNaam.Value)                     && Bestaat het bedrijf?
   ? Accounts->Bedrijf                           && Zo ja, naam ervan weergeven
ELSE                                             && Zo nee, overeenkomstige namen zoeken
   SET ORDER TO SoundComp                        && SOUNDEX()-index gebruiken
   IF SEEK(SOUNDEX(Form.BedrNaam.Value))         && Zijn er gelijkluidende namen?
      SELECT (Accounts)                          && Zo ja, deze namen weergeven
      LIST BedrNaam WHILE ;
         DIFFERENCE(Bedrijf,Form.BedrNaam.Value) >2
   ELSE                                          && Geen gelijkluidende namen
      ? "Er zijn geen bedrijfsnamen die lijken op de naam die u zoekt."
   ENDIF
ENDIF
```

Tekenreeksen aaneenschakelen

Het is mogelijk dat u bij het samenstellen van tekenuitdrukkingen twee of meer reeksen moet vergelijken. Voeg reeksen aan elkaar toe (schakel reeksen aaneen) met de plusteken-operator (+) of de minteken-operator (–).

- Met de plusteken-operator wordt de reeks rechts van de operator toegevoegd aan het einde van de reeks links van de operator. Dit is de meest gebruikelijke methode voor aaneenschakeling.
- Met de minteken-operator worden spaties vanaf het einde van de linkerreeks verwijderd voordat de rechterreeks wordt toegevoegd. Vervolgens worden de spaties toegevoegd aan het einde van de rechterreeks. Deze methode is bijzonder geschikt voor het uitlijnen van kolommen met informatie.

```
? "Hallo "+ "Jan."        && Geeft "Hallo Jan." als resultaat
? "Hallo "- "Jan."        && Geeft "HalloJan. " als resultaat
```

In het volgende voorbeeld worden zowel de plusteken- als de minteken-operator gebruikt voor het weergeven van de plaats, regio en postcode. Met de minteken-

operator worden de spaties die worden gebruikt voor de uitvulling van het veld PLAATS, verplaatst naar de rechterkant van veld AREACODE. Hierdoor worden de postcodes verticaal uitgelijnd.

```
USE Afnemers
LIST OFF ALL PLAATS - ", "+ AREACODE+ " " + ZIPCODE
```

De plaatsen worden als volgt weergegeven:

```
Atlanta,          GA          30329
San Jose,         CA          95131
Wilmette          IL          60091
Nashville,        TN          37215
New York,         NY          10023
```

Een deel van een tekenreeks ophalen

Het is mogelijk dat u bij het samenstellen van tekenuitdrukkingen een deel van een reeks wilt ophalen. U wilt bijvoorbeeld de voornaam ophalen uit een veld NAAM. Een reeks die u ophaalt uit een andere reeks wordt een *subreeks genoemd*. dBASE bevat de volgende functies voor het ophalen van subreeksen uit een reeks:

- LEFT() haalt links van een reeks een subreeks op
- RIGHT() haalt rechts van een reeks een subreeks op
- SUBSTR() haalt een subreeks op uit een willekeurige positie in een reeks

Dit zijn enkele voorbeelden van het ophalen van subreeksen:

```
mRegio = "Californië"
mPlaats = "Scotts Valley"
mPostcode =  "95066"
? UPPER(LEFT(mRegio,1)+RIGHT(mRegio,1))              && Geeft Cë als resultaat
? SUBSTR(mPostcode,1,2)                              && Geeft 95 als resultaat
Reverse = SUBSTR(mPlaats,8,6) + " " + SUBSTR(mPlaats,1,6)
? Reverse                                            && Geeft Valley Scotts als
                                                     && resultaat
```

Als u subreeksen ophaalt, moet u in sommige gevallen de positie bepalen van de subreeks in de reeks. dBASE beschikt daartoe over de volgende functies:

- AT() geeft de startpositie van een subreeks in een reeks als resultaat, waarbij van links naar rechts wordt gezocht en geteld
- RAT() geeft de startpositie van een subreeks in een reeks als resultaat, waarbij van rechts naar links wordt gezocht en van links naar rechts wordt geteld

AT() en RAT() geven 0 als resultaat als de subreeks niet in de reeks wordt aangetroffen.

U kunt een subreeks ophalen zonder dat u de positie van de subreeks in de hoofdreeks kent. In het volgende voorbeeld worden de functies AT(), LEFT() en SUBSTR() gebruikt om de elementen af te drukken in een door komma's gescheiden lijst. In de code wordt de komma gebruikt als een markering om te zoeken naar de elementen in de lijst en deze vervolgens weer te geven.

```
mBestandenLijst = "Acct_rec,Klant,Werknmer,Orders,Verkoper"
DO WHILE "," $ mBestandenLijst                          && Als lijst komma bevat
   ? LEFT(mBestandenLijst, AT(",",mBestandenLijst)-1)   && Eerste woord weergeven
   mBestandenLijst = SUBSTR(mBestandenLijst, AT(",",mBestandenLijst)+1)
                                                        && Eerste woord verwijderen
ENDDO
? mBestandenLijst                                       && Laaste woord weergeven
```

Hoofdlettergebruik wijzigen of bepalen

U kunt het hoofdlettergebruik in een tekenreeks met verschillende functies testen of wijzigen. Een reeks kan enkel *HOOFDLETTERS* of enkel *kleine letters* bevatten. Het is ook mogelijk dat de *Eerste Letter* van elk woord in de reeks een hoofdletter is.

- UPPER() zet een reeks om in enkel hoofdletters. ISUPPER() geeft True (waar) als resultaat als het eerste teken van een reeks een hoofdletter is.
- LOWER() zet een reeks om in enkel kleine letters. ISLOWER() geeft True (waar) als resultaat als het eerste teken van een reeks een kleine letter is.
- PROPER() zet de eerste letter van elk woord in de reeks om in een hoofdletter. Gebruik PROPER() voor reeksen die bepaalde personen, plaatsen of zaken, zoals "Vincent", "Den Helder", Pasen", aanduiden.

Me de code in het volgende voorbeeld wordt de inhoud van het veld REGIO (twee tekens) in de tabel Klanten vervangen door hoofdletters.

```
USE Klanten
REPLACE ALL Regio WITH UPPER(Regio)
```

Met de code in het volgende voorbeeld wordt in het veld NAAM in de tabel Landen de eerste letter van het land omgezet in een hoofdletter. De resterende letters worden omgezet in kleine letters.

```
USE Landen
REPLACE ALL Naam WITH UPPER(LEFT(Naam,1))+LOWER(SUBSTR(Naam,2,10))
```

In de voorgaande code worden de hoofdletters alleen correct gebruikt als de landnaam uit één woord bestaat. Een landnaam als "Nieuw-Zeeland" wordt door de code gewijzigd in "Nieuw-zeeland". De onderstaande code is een betere methode om de eerste letters van woorden om te zetten in hoofdletters. Hiertoe gebruikt u PROPER():

```
USE Landen
REPLACE ALL Naam WITH PROPER(Naam)
```

In het Nederlands variëren kleine letters van a tot z en hoofdletters van A to Z. In andere talen wordt soms een ander onderscheid gemaakt tussen hoofdletters en kleine letters. Het huidige taalaansturingsprogramma bepaalt hoe dBASE de hoofdletters of kleine letters in een reeks beschouwt. Zie Appendix C voor meer informatie.

Spaties verwijderen

Soms is het nodig dat u extra spaties aan het begin of einde van een reeks verwijdert.

- TRIM() en RTRIM() zijn identieke functies waarmee u de spaties rechts van een reeks verwijdert
- LTRIM() is een functie waarmee u de spaties links van een reeks verwijdert

De velden in een tabel of de invoervakken in een formulier hebben een vaste lengte die is afgestemd op de maximumlengte van een reeks. Als u een bepaalde reeks weergeeft, kunt u de extra spaties aan de rechterkant verwijderen, zodat u twee reeksen naast elkaar kunt weergeven. In dit voorbeeld worden de spaties in het veld PLAATS verwijderd zodat een adres kan worden weergegeven:

```
? TRIM(Plaats) + ", " + Regio + " "+Postcode     && Geeft "Dallas, TX 75207" weer
```

In de sectie "Tekenreeksen aaneenschakelen", eerder in dit hoofdstuk, wordt een andere techniek beschreven voor het verwijderen van spaties, namelijk het gebruik van de minteken-operator.

Tekens herhalen

dBASE biedt twee functies voor het herhalen van teken in een reeks, zodat u deze tekens niet handmatig hoeft te typen.

- SPACE() herhaalt een spatie het opgegeven aantal keren. Dit is handig als u tekenvariabelen wilt initialiseren.
- REPLICATE() herhaalt het opgegeven teken het opgegeven aantal keren. Dit is handig als u gebruik wilt maken van visuele effecten, zoals een rij onderstreeptekens of asterisken.

```
mNaam = SPACE(30)                 && Variabele met 30 spaties initialiseren
mBeroep = SPACE(45)               && Variabele met 45 spaties initialiseren
mSterRegel = REPLICATE("*",30)    && Rij van 30 sterretjes maken
? mStarline                       && Geeft ****************************** weer
```

Werken met gegevens van het type Numeriek

Gegevens van het type Numeriek zijn gegevens waarop u wiskundige bewerkingen uitvoert, zoals optellen of vermenigvuldigen, of die u gebruikt in wiskundige functies.

In dBASE IV kunt u numerieke gegevens in twee veldtypen opgeven (Numeriek en Zwevend) en kunt u deze gegevens op verschillende manieren verwerken. In dBASE IV hebben waarden van het type Numeriek een maximumprecisie van 20 cijfers en hebben waarden van het type Zwevend een maximumprecisie van 15. Hoewel Zwevend in dBASE voor Windows is gehandhaafd als *veldtype*, is er geen verschil tussen waarden van het type Numeriek en waarden van het type Zwevend. In wiskundige berekeningen biedt dBASE dezelfde precisie (19 cijfers) voor beide typen. Het veldtype Zwevend is gehandhaafd vanwege compatibiliteit met dBASE IV.

Als u een getal toewijst aan een geheugenvariabele, geeft de functie TYPE() altijd N als resultaat. TYPE() geeft alleen F als resultaat voor velden van het type Zwevend.

In dBASE IV geeft u met SET PRECISION het aantal cijfers op dat intern wordt gebruikt voor wiskundige berekeningen. In dBASE voor Windows heeft SET PRECISION

betrekking op *vergelijkingen van gegevens, maar niet op wiskundige berekeningen*. In Afbeelding 6.1 wordt aangegeven hoe de instelling van de precisie van invloed is op numerieke bewerkingen in dBASE voor Windows en dBASE IV:

Afbeelding 6.1 Vergelijking van SET PRECISION in dBASE voor Windows en dBASE IV

dBASE voor Windows

```
* Deze code geeft dezelfde resultaten als in dBASE IV
SET DECIMALS TO 18
SET PRECISION TO 19
x=0.123456789012345678
y=0.123456789012345688
? x=y     && Geeft .F. als resultaat
? x-y     && Geeft 0,246913578024691350 als
          && resultaat

* Deze resultaten verschillen van dBASE IV
SET PRECISION TO 16
? x=y     && Geeft nu .T. als resultaat
? x-y     && Geeft nog steeds 0,246913578024691350 als
          && resultaat
```

dBASE IV

```
SET DECIMALS TO 18
SET PRECISION TO 19
x=0.123456789012345678
y=0.123456789012345688
? x=y     && Geeft .F. als resultaat
? x-y     && Geeft 0,246913578024691350 als
          && resultaat


SET PRECISION TO 16
? x=y     && Geeft nog steeds .F. als resultaat
? x-y     && Geeft nu 0,246913578024691400 als
          && resultaat
```

In Tabel 6.3 worden enkele algemene numerieke functies in dBASE voor Windows opgesomd.

Tabel 6.3 Algemene numerieke functies

Functie	Geeft als resultaat
ABS()	De absolute waarde van een getal (verwijdert het algebraïsche teken)
EXP()	Het grondtal e verheven tot de opgegeven macht (exponent)
LOG()	De natuurlijke logaritme verheven tot het grondtal e van een getal
LOG10()	De logaritme van een getal (met grondtal 10)
MAX()	Het grootste getal van twee getallen
MIN()	Het kleinste getal van twee getallen
MOD()	De rest (modulus) van de deling van een getal door een ander getal
PI()	De waarde van pi, bij benadering (de verhouding van een cirkelomtrek tot de cirkeldiameter)
RANDOM()	Een willekeurig getal
SIGN()	Het teken (positief of negatief) van een getal
SQRT()	De vierkantswortel van een getal

In Hoofdstuk 20 worden CALCULATE, SUM en andere commando's beschreven waarmee u numerieke gegevens in tabellen samenvat.

Globale weergave-instellingen voor getallen opgeven

Verscheidene SET-commando's hebben invloed op de opmaak van getallen die worden weergegeven of naar de printer worden gestuurd. De waarde van getallen wordt door

deze instellingen niet gewijzigd. Alleen de manier waarop getallen worden weergegeven, wordt gewijzigd. Als u een SET-commando hebt gebruikt, wordt de opgegeven opmaak daarna gebruikt voor alle commando's waarmee u getallen weergeeft, zoals ?, DEFINE ENTRYFIELD en BROWSE. In Tabel 6.4 wordt een overzicht gegeven van de SET-commando's waarmee u de opmaak voor getallen instelt.

Tabel 6.4 Weergave-instellingen voor getallen

SET-commando	Weergave-instelling
SET CURRENCY	Geeft op of het valutasymbool links of rechts van het getal wordt weergegeven als het opmaaksymbool "$" wordt gebruikt
SET CURRENCY TO	Geeft op welk teken wordt gebruikt voor het valutasymbool als het opmaaksymbool "$" wordt gebruikt
SET DECIMALS	Geeft het aantal decimalen op dat wordt weergegeven. SET DECIMALS heeft geen invloed op de interne precisie van het getal
SET POINT	Geeft het teken op waarmee decimale cijfers worden gescheiden van het hele getal
SET SEPARATOR	Geeft het teken op waarmee elke groep van drie cijfers wordt gescheiden in een waarde die groter is dan 999

In de volgende voorbeelden ziet u enkele numerieke weergave-instellingen voor de gulden.

```
Geld = 543567.4532
SET CURRENCY LEFT                    && Valutasymbool aan rechterkant weergeven
SET CURRENCY TO "F"                  && Guldens
SET DECIMALS TO 2
SET POINT TO ","
SET SEPARATOR TO "."
? Geld                               && Geeft 543567,45 als resultaat
? Geld Picture "999,999.99"          && Geeft 543.567,45 als resultaat
? Geld Picture "@$999,999.99"        && Geeft F 543567,45 als resultaat
? Geld Function "$"                  && Geeft F 543567,45 als resultaat
```

Getallen weergeven in formulieren

Numerieke waarden die zijn opgeslagen in een veld of een variabele, worden in een formulier weergegeven wanneer u een stuurelement maakt, zoals een invoervak, tekstvak of ringveld, en een numeriek veld of een numerieke variabele aan het stuurelement koppelt. Elk van deze stuurelementen heeft twee speciale kenmerken waarmee u kunt instellen hoe de gegevens worden weergegeven.

- Met het kenmerk Function geeft u opmaak- of invoerbeperkingen op voor de hele variabele of het hele veld
- Met het kenmerk Picture geeft u opmaak- of invoerbeperkingen op voor een deel van de variabele of het veld

In Tabel 6.5 ziet u enkele voorbeelden van de manier waarop u gegevens van het type Numeriek kunt opmaken met de kenmerken Function en Picture.

Tabel 6.5 Getallen opmaken met de kenmerken Function en Picture

Opmaak	Instelling van kenmerk
Geef een getal weer met exponentiële notatie	Function "^"
Beperk de weergave tot cijfers, tekens en spaties	Picture "999" of Picture "###" (feitelijk aantal cijfers of hekjes (#) varieert)
Geef nulwaarden weer als een lege reeks	Function "Z"
Geef voorloopnullen weer als nullen, dollartekens of asterisken	Function "L"
Geef een getal weer met de standaardopmaak voor valuta	Function "$"
Geef CR (credit) weer na een positief getal en DB (debet) na een negatief getal	Function "C", Function "X"
Geef negatieve getallen tussen haakjes weer	Function "("
Centreer een getal, lijn een getal rechts uit of lijn een getal links uit	Function "I", Function "J", Function "B"

In de volgende code wordt een invoervak ingesteld op de weergave van de naam van een persoon. Met de instelling van het kenmerk Picture wordt de gegevensinvoer beperkt tot letters. Met de instelling van het kenmerk Function worden voorloop- en volgspaties verwijderd.

```
DEFINE ENTRYFIELD Gewicht OF This PROPERTY;
   Width 3, Top 11, Left 21, Height 1, ColorNormal "bg+ b", ;
   Border .F., Picture "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX", Function "T", ;
   DataLink "Diagnose"
```

Financiële berekeningen uitvoeren

dBASE bevat drie functies voor berekeningen met rente uit investeringen of leningen.

- FV() berekent de toekomstige waarde van een lening of investering op basis van gelijke periodieke betalingen tegen een vast rentetarief
- PAYMENT() berekent het periodieke bedrag dat nodig is voor de afbetaling van een lening of investering op basis van een opgegeven hoofdsom, termijnbedrag en rentetarief
- PV() berekent de huidige waarde van een lening of investering op basis van gelijke periodieke betalingen tegen een vast rentetarief

Een investering van bijvoorbeeld F 250 per maand met een rentetarief van 7 procent op jaarbasis die maandelijks wordt berekend over een periode van 10 jaar, heeft de volgende eindwaarde:

```
mInvest = FV(250,.07/12,12*10)
? mInvest Function "$"              && Geeft F 43271,20 als resultaat
```

Aangezien toekomstige waarde het totaal aan deposito's en gegenereerde rente weergeeft, kunt u het totaalbedrag aan rente na 10 jaar als volgt berekenen:

```
? mInvest - (250*120)                    && Geeft $13271,20 als resultaat
```

Als u de maximale hypotheek wilt berekenen op basis van een bepaald aflossingsbedrag en heersende rentetarieven, gebruikt u PV(). Stel dat iemand een huis wil kopen en F 700 per maand wil aflossen. Als het rentetarief voor een vaste hypotheek met een looptijd van 30 jaar momenteel 7 procent bedraagt, gebruikt u de volgende code:

```
mHgteLening = PV(700,.07/12,12*30)
? mHgteLening Function "$"               && Geeft F 105215,30 als resultaat
```

Gebruik PAYMENT() als de hoofdsom bekend is en u de hoogte van de periodieke betaling wilt berekenen. Gebruik de volgende code als u de maandelijkse aflossing wilt berekenen voor een hypotheek van F 100.000 tegen een rente van 7 procent, met een looptijd van 30 jaar:

```
mHypotAfl = PAYMENT(100000,.07/12,12*30)
? mHypotAfl Function "$"                 && Geeft F 665,30 als resultaat
```

Trigonometrische berekeningen uitvoeren

In technische en wetenschappelijke applicaties moeten vaak trigonometrische berekeningen worden uitgevoerd. In Tabel 6.6 wordt een overzicht gegeven van de trigonometrische functies die beschikbaar zijn in dBASE. De trigonometrische functies geven waarden van het type Zwevend als resultaat.

Tabel 6.6 Trigonometrische functies

Functie	Geeft als resultaat
ACOS()	De boogcosinus (omgekeerde cosinus) van een getal
ASIN()	De boogsinus (omgekeerde sinus) van een getal
ATAN()	De boogtangens (omgekeerde tangens) van een getal
ATN2()	De boogtangens (omgekeerde tangens) van een punt
COS()	De cosinus van een hoek
DTOR()	De hoekwaarde in radialen, gemeten in graden
RTOD()	De hoekwaarde in graden, gemeten in radialen
SIN()	De sinus van een hoek
TAN()	De tangens van een hoek

In het volgende voorbeeld wordt de lengte berekend van een dakspant voor een kamer met een breedte van 12,19 meter. De spant heeft een hoek van 30 graden en een overhang van 45,7 cm (0,457 m):

```
mSpant = (12.19/2)/COS(DTOR(30)) + 0.457
? mSpant                                 && Geeft 7,49 als resultaat
```

Getallen afkappen en afronden

Numerieke uitdrukkingen geven vaak decimale waarden als resultaat. Als u voor een berekening een resultaat met gehele getallen (zonder decimalen) nodig hebt, moet u de

decimalen afronden of afkappen. dBASE bevat verschillende functies voor het afkappen en afronden van getallen:

- CEILING() geeft als resultaat het dichtstbijzijnde gehele getal dat *groter* dan of gelijk is aan een opgegeven getal
- FLOOR() geeft als resultaat het dichtstbijzijnde gehele getal dat *kleiner* dan of gelijk is aan een opgegeven getal
- INT() geeft als resultaat het deel voor de komma van een opgegeven getal, waarbij de decimalen worden afgekapt
- ROUND() rondt een opgegeven getal af op een opgegeven aantal decimalen

Het volgende voorbeeld is gebaseerd op het voorbeeld met de dakspant uit de vorige paragraaf:

```
? mSpant                    && Geeft 8,49 als resultaat
? CEILING(mSpant)           && Geeft 9,00 als resultaat
? FLOOR(mSpant)             && Geeft 8,00 als resultaat
? ROUND(mSpant,1)           && Geeft 8,50 als resultaat
? INT(mSpant)               && Geeft 8 als resultaat
? mSpant - INT(mSpant)      && Geeft 0,49, het decimale gedeelte, als resultaat
SET DECIMALS TO 8           && De standaardinstelling is 2 decimalen
? mSpant                    && Geeft 8,49489978 als resultaat
? ROUND(mSpant,1)           && Geeft 8,50000000 als resultaat
SET DECIMALS TO 2
? mSpant                    && Geeft 8,49 als resultaat
```

Werken met gegevens van het type Datum

Velden en variabelen van het type Datum bevatten waarden die kalenderdatums weergeven. Datumwaarden hebben sommige karakteristieken met getallen gemeen. Twee datums kunt u net als getallen vergelijken als u wilt weten welke datum eerder valt (kleiner is) of later valt (groter is). Bovendien kunt u een aantal dagen aan een datum toevoegen of ervan aftrekken, waarna de resulterende datum wordt berekend. Op dezelfde wijze kunt u de ene datum van de andere aftrekken als u het aantal tussenliggende dagen wilt weten.

Met de volgende opdrachten bepaalt u een datum die 90 dagen voor de vervaldatum van een rekening valt:

```
mVervaldat = {01-01-95}         && Kan een veldwaarde zijn
mVervaldat90 = mVervaldat - 90
? mVervaldat90                  && Geeft 03-10-94 als resultaat
```

U kunt ook elementaire tijdwaarden opslaan met datums door gedeelten van dagen toe te voegen of af te trekken. Bijvoorbeeld:

```
mInDatum = {01-07-1988}         && 01-07-88
mUitDatum = mInDatum + .25      && 01-07-88
? mInDatum = mUitDatum          && .F.
```

Hoewel *mInDatum* en *mUitDatum* op dezelfde dag vallen, wordt intern een waarde opgeslagen voor *mUitDatum* die 1/4 dag groter is dan de waarde van *mInDdatum*. Als u niet wilt dat verschillen van minder dan een dag worden weergegeven in een vergelijking, converteert u de gegevens van het type Datum naar gegevens van het type Teken. Hiertoe gaat u als volgt te werk:

```
? DTOC(mInDatum) = DTOC(mUitDatum)      && Geeft .T. als resultaat
```

In het volgende voorbeeld wordt de variabele *mVersch* geïnitialiseerd met het verschil tussen de huidige datum en een transactiedatum. Er wordt een herinneringsbrief gemaakt als de vervaldatum van de rekening met 30 tot 45 dagen is overschreden. Als de vervaldatum met meer dan 45 dagen is overschreden, wordt een aanmaning gemaakt met ferme bewoordingen:

```
mVersch = DATE() - Transdat                    && Transdat is de transactiedatum
DO CASE
   CASE mVersch > 30 .AND. mVersch <= 45       && Transactiedatum tussen 30 en 45 dagen
                                               && geleden
      DO BeleefdeHerinnering
   CASE mVersch > 45                           && Transactiedatum meer dan 45 dagen geleden
      DO OnmiddellijkBetalen
ENDCASE
```

Datums opmaken en weergeven

U kunt datums in verschillende notaties weergeven. De instellingen voor de notatie die u opgeeft met de optie Internationaal van Configuratiescherm, worden als standaardinstellingen gebruikt. Bovendien kunt u een gedeelte van een datum, zoals de dag van de week, ophalen en er een speciale notatie op toepassen.

In Tabel 6.7 wordt een overzicht gegeven van de commando's en functies voor datums:

Tabel 6.7 Commando's en functies voor datums

Commando of functie	Beschrijving
CDOW(), DOW()	Geeft de dag van de week als resultaat, respectievelijk in de vorm van een woord en een getal. Zondag wordt beschouwd als de eerste dag van de week
CMONTH(), MONTH()	Geeft de maand als resultaat, respectievelijk in de vorm van een woord en een getal
DAY()	Geeft de dag van de maand als resultaat in de vorm van een getal
DMY(), MDY()	Geeft de datum als resultaat met respectievelijk de notatie DD Maand JJ en Maand DD, JJ
SET CENTURY	Geeft het aantal cijfers op (2 of 4) in het jaargedeelte van een datum
SET DATE	Geeft de algemene datumnotatie op (MM-DD-JJ, DD-MM-JJ, enz.) volgens de conventies die in een bepaald land gelden
SET DATE TO	Stelt de systeemdatum in
SET MARK	Geeft het teken op waarmee het jaar, de maand en de dag in een datum worden gescheiden
YEAR()	Geeft het jaar van een datum als resultaat in de vorm van een getal met vier cijfers

In de volgende voorbeelden ziet u hoe het opmaken en weergeven van een datum in zijn werk gaat:

```
mDatum = {01-06-94}
SET CENTURY OFF          && Jaar weergeven met twee cijfers
? CDOW(mDatum)           && Geeft woensdag als resultaat
? DOW(mDatum)            && Geeft 4 als resultaat
? CMONTH(mDatum)         && Geeft juni als resultaat
? MONTH(mDatum)          && Geeft 6 als resultaat
? DAY(mDatum)            && Geeft 1 als resultaat
? DMY(mDatum)            && Geeft 1 juni 1994 als resultaat
? MDY(mDatum)            && Geeft juni 01, 1994 als resultaat
SET CENTURY ON           && Geef het jaar weer met vier cijfers
? mDatum                 && Geeft 01-06-1994 als resultaat
? MDY(mDatum)            && Geeft juni 01, 1994 als resultaat
SET DATE JAPAN
? mDatum                 && Geeft 1994/06/01 als resultaat
SET DATE ITALIAN
? mDatum                 && Geeft 01-06-1994 als resultaat
SET MARK TO "*"
? mDatum                 && Geeft 01*06*1994 als resultaat
SET CENTURY OFF
? YEAR(mDatum)           && Geeft 1994 als resultaat (YEAR() negeert SET CENTURY)
```

Werken met tijdgegevens

Hoewel het gegevenstype tijd niet bestaat, kunt u tijdgegevens manipuleren en opslaan als tekenreeksen. De functie TIME() geeft de huidige tijd terug als een tekenreeks. Op de volgende wijze slaat u de huidige tijd op in een geheugenvariabele:

```
mNu = TIME()          && Geeft een reeks zoals "14:30:43" als resultaat
```

Vervolgens kunt u *mNu* splitsen in subreeksen als u enkel een gedeelte van de tijd wilt gebruiken. Als de tijd bijvoorbeeld 14:30 en 43 seconden is, plaatst u de reeks 14:30 met de volgende instructie in *mNu*.

```
mNu = LEFT(mNu,5)     && Geeft "14:30" als resultaat
```

In Tabel 6.8 wordt een overzicht gegeven van de commando's en functies die u gebruikt voor tijdgegevens.

Tabel 6.8 Commando's en functies voor tijdgegevens

Commando of functie	Beschrijving
ELAPSED()	Geeft het aantal seconden tussen twee opgegeven tijdstippen als resultaat
SECONDS()	Geeft het aantal seconden als resultaat dat is verlopen sinds 24:00 (middernacht)
SET TIME	Stelt de systeemtijd in
TIME()	Geeft de systeemtijd als resultaat

In de volgende code worden de tijdfuncties gebruikt voor een berekening van de tijd die nodig is voor het uitvoeren van een subroutine:

```
mBegin = TIME()                          && Huidige tijd opslaan (andere code?)
DO LangeRoutine
mEind = TIME()                           && Tijd opslaan na uitvoeren van enige code
mDuur = ELAPSED(mEind,mBegin) * 3600     && Seconden omzetten in uren
? mDuur                                  && Aantal verstreken uren weergeven
```

Zie bladzijde 216 voor een voorbeeld van de validatie van tijdgegevens in een formulier.

Werken met gegevens van het type Logisch

Gegevens van het type Logisch, ook wel Booleaanse gegevens genoemd, kunnen maar twee waarden hebben: True (.T.) of False (.F.). In logische velden en variabelen worden doorgaans de schuine streep (/) of de gegevens zelf opgeslagen, zoals Gehuwd, Betaald of Vrijgesteld. In formulieren plaatst u stuurelementen van het type aankruisvakje als u logische waarden wilt weergeven en bewerken.

Als u logische uitdrukkingen maakt, hoeft u de uitdrukkingen niet te beëindigen met "= .T." of "= .F.". Een logische waarde *is* een uitdrukking. U kunt gewoon FOR Exempt gebruiken in plaats van FOR Exempt = .T., hoewel beide uitdrukkingen geldig zijn.

U kunt twee logische uitdrukkingen, zoals FOR Exempt = Status, vergelijken. De uitdrukking wordt geëvalueerd en op basis van de vergelijking wordt een andere logische waarde (.T. of .F.) als resultaat gegeven.

In dBASE kunt u T (True, waar) of Y (yes, ja) gebruiken als een positieve waarde en F (False, onwaar) of N (No, nee) als een negatieve.

In de volgende routine wordt de logische operator .NOT. samen met de functies FOUND() en EOF() gebruikt om de mogelijke resultaten van een zoekbewerking met SET NEAR ON te verwerken.

```
USE Levrncrs ORDER Verkopernr
SET NEAR ON              && Plaatst de recordaanwijzer nabij een waarde die wordt gezocht
                         && maar nog niet is gevonden
mSleutel = Verkopernr
SEEK mSleutel
DO CASE
   CASE FOUND()          && Exacte overeenkomst gevonden, gegevens weergeven
      ? Verkopernr, Verkoper, Telefoon, Plaats
   CASE .NOT. EOF()      && Verkoper niet gevonden, maar wel een soortgelijke
      ? "Verkoper niet gevonden"
      ? "Volgende verkoper wordt weergegeven (in alfabetische volgorde)"
      ? Verkopernr, Verkoper, Telefoon, Plaats
   OTHERWISE             && Verkoper niet gevonden, en in het bestand zijn geen
                         && soortgelijke namen aanwezig
      ? "Verkoper niet gevonden"
      ? "Recordaanwijzer onderaan het bestand"
ENDCASE
```

Werken met niet-dBASE-gegevenstypen

Niet-dBASE-tabellen, zoals Paradox-tabellen, bevatten mogelijk velden met gegevenstypen die niet voorkomen in Tabel 6.1. Met dBASE kunt u deze velden wel lezen en ernaar schrijven. Als u gegevens uit een niet-dBASE-veldtype opslaat in een geheugenvariabele, worden de gegevens geconverteerd naar een gegevenstype dat wel wordt ondersteund. En als u een niet-dBASE-veldtype in een geheugenvariabele vervangt, worden de gegevens geconverteerd naar het niet-dBASE-veldtype.

Paradox-tabellen ondersteunen bijvoorbeeld valutavelden voor de opslag van geldwaarden. Als u het valutaveld toewijst aan een geheugenvariabele, wordt een numerieke variabele gemaakt. Op dezelfde wijze kunt u een valutaveld vervangen door de inhoud van een numerieke variabele.

In Hoofdstuk 23 wordt beschreven hoe Paradox-gegevenstypen door dBASE worden gebruikt.

Als u een niet-dBASE-subroutine gebruikt, zoals een C-functie uit een DLL (Dynamic Link Library), worden de gegevenstypen van parameters of resultaatwaarden geconverteerd op basis van een prototype dat u definieert met het commando EXTERN. Zie Hoofdstuk 25 voor meer informatie.

Gegevenstypen converteren

Soms moet u gegevens converteren van het ene type naar het andere. Als u bijvoorbeeld tekst wilt weergeven die zowel tekenreeksen als een datum of getal bevat, converteert u de gegevens die niet van het type Teken zijn naar dit type. En bij berekeningen met verschillende gegevenstypen moet u alle gegevens converteren naar hetzelfde type.

Tabel 6.9 bevat een overzicht van de functies voor de conversie van gegevenstypen.

Tabel 6.9 Gegevenstypen converteren

Conversie	Functie
Teken naar Numeriek	VAL()
Numeriek naar Teken	STR()
Numeriek naar Zwevend	FLOAT()
Teken naar Datum	CTOD()
Datum naar Teken	DTOC()
Datum naar Teken (voor het maken van een index); geeft een reeks met de notatie JJJJMMDD als resultaat	DTOS()
Numeriek, Logisch of Datum naar Teken	TRANSFORM()

De volgende voorbeelden bevatten enkele standaardconversies van gegevenstypen.

Als u een index wilt maken op basis van besteldatum en bestelde hoeveelheid, converteert u zowel de datumvelden als de numerieke velden naar tekenreeksen:

```
INDEX ON DTOS(Besteldat) + STR(Aantal,8,2) TO Datum_Aantal
```

In de volgende code worden de twee tekens voor maand en jaar opgehaald uit de datum (uitgaande van de datumnotatie MM/DD/JJ). Vervolgens worden deze tekens aaneengeschakeld met het klantnummer om een uniek factuurnummer te genereren:

```
mDit_jaar = RIGHT(DTOC(DATE()),2)
mDeze_maand = LEFT(DTOC(DATE()),2)
mFactuurnr = mKlantnr + mDit_jaar + mDeze_maand
```

In het volgende voorbeeld neemt m_Nr, een identificatiecode voor werknemers (bijvoorbeeld B1234), toe met één. Hiertoe worden de vier cijfers geconverteerd naar numerieke gegevens met de functie VAL() en wordt één opgeteld bij het resulterende getal. Vervolgens wordt het getal met behulp van STR() teruggeconverteerd naar een tekenreeks en wordt het resultaat aaneengeschakeld met de beginletter van de identificatiecode.

```
m_Nr = LEFT(m_Nr,1) + TRANSFORM(VAL(RIGHT(m_Nr,4)) + 1,"@L 9999")
```

Hoofdstuk

7

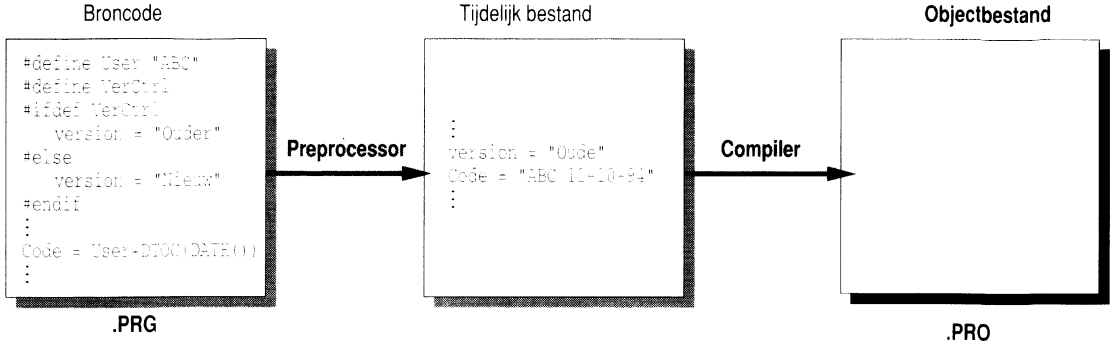
Werken met preprocessor-instructies

Preprocessor-instructies zijn opdrachten die u in dBASE-code opneemt en waarmee u het compileerprogramma opdracht geeft code uit te voeren voordat deze wordt gecompileerd. Met preprocessor-instructies kunt u bijvoorbeeld "zoek-en-vervang"-bewerkingen uitvoeren op tekst, kunt u voorwaardelijke compilatie instellen of kunt u compilatie-opties opgeven. Telkens wanneer u code compileert, wordt in dBASE automatisch gezocht naar preprocessor-instructies en worden deze geëvalueerd. Doordat u toegang hebt tot de preprocessor, hebt u meer controle over het compilatieproces en kunt u de prestaties van uw applicaties verbeteren. In dit hoofdstuk wordt een inleiding gegeven tot de preprocessor van dBASE en wordt beschreven hoe u preprocessor-instructies gebruikt.

Over het gebruik van de preprocessor

In dBASE wordt een single-pass compileerprogramma gebruikt met een ingebouwde preprocessor. Als u een programmabestand compileert, bijvoorbeeld met het commando COMPILE, wordt in de code gezocht naar preprocessor-instructies en worden deze geëvalueerd, waarbij een tijdelijk tussenbestand wordt gegenereerd dat door het compileerprogramma wordt gecompileerd. (Gebruik SET DEVELOPMENT als u wilt bepalen of programma's automatisch worden gecompileerd nadat de broncode is gewijzigd.) In Afbeelding 7.1 wordt een overzicht gegeven van het compilatieproces.

Afbeelding 7.1 Preprocessor-instructies verwerken en programma compileren



Elke instructie begint met een hekje (#). Elke coderegel die begint met een hekje, wordt beschouwd als een preprocessor-instructie. Het hekje (#) kan worden voorafgegaan door spaties. Tabel 7.1 bevat een overzicht van de preprocessor-instructies.

Tabel 7.1 Overzicht van preprocessor-instructies in dBASE

Instructie	Beschrijving
#define	Definieert een identificatiesymbool
#if ... #endif	Compileert een deel van de code als een identificatiesymbool een bepaalde waarde heeft
#ifdef ... #endif	Compileert een deel van de code als een identificatiesymbool is gedefinieerd (met #define)
#ifndef ... #endif	Compileert een deel van de code als een identificatiesymbool niet is gedefinieerd
#include	Voegt een broncodebestand (ook wel *header-bestand* genoemd) in op de huidige positie
#pragma	Geeft een compilatie-optie op
#undef	Maakt de definitie van een identificatiesymbool ongedaan

Constanten definiëren

U kunt de prestaties van een programma op een eenvoudige manier verbeteren door constanten aan te duiden met *identificatiesymbolen*. Een identificatiesymbool is een beschrijvende naam die u aan programmatekst toekent met de instructie #define. De voordelen van het gebruik van identificatiesymbolen worden duidelijk als u nagaat op welke andere manieren u constanten in dBASE kunt weergeven: met letterlijke waarden en met geheugenvariabelen. In deze paragraaf worden deze drie manieren om constanten te definiëren, met elkaar vergeleken.

Letterlijke waarden gebruiken als constanten

In het volgende voorbeeld wordt de letterlijke waarde 5000 gebruikt om het maximumaantal klanten weer te geven.

```
DO WHILE nHuidigeKlant < 5000
   :
   nHuidigeKlant = nHuidigeKlant + 1
ENDDO
```

Het gebruik van letterlijke waarden is het eenvoudigst, maar heeft een nadeel. Als u het maximumaantal klanten wilt verhogen, moet u het hele programma doorzoeken en 5000 wijzigen in de nieuwe waarde. U kunt geen globale zoek- en vervangbewerking uitvoeren, aangezien 5000 niet overal hoeft te verwijzen naar het maximumaantal klanten. Om een en ander gemakkelijker te maken, kunt u een geheugenvariabele gebruiken om de constante waarde weer te geven.

Geheugenvariabelen gebruiken als constanten

Als u constante waarden weergeeft met geheugenvariabelen, is uw code gemakkelijker te begrijpen. Als u daarnaast de waarde van een constante wilt wijzigen (bijvoorbeeld om het maximumaantal klanten te verhogen), hoeft u alleen een wijziging aan te brengen in de regel waarmee de variabele wordt geïnitialiseerd. In het volgende voorbeeld wordt een geheugenvariabele in plaats van een constante gebruikt om het maximumaantal klanten weer te geven:

```
nMaxKlant = 5000                    && nMaxKlant initialiseren
DO WHILE nHuidigeKlant < nMaxKlant  && nMaxKlant gebruiken als constante
   :
   nHuidigeKlant = nHuidigeKlant + 1
ENDDO
```

Als u een geheugenvariabele gebruikt, lopen de prestaties echter enigszins terug omdat de variabele voortdurend moet worden geïnitialiseerd. In dit geval gebeurt dat via de DO WHILE-lus. Bovendien gebruikt elke variabele die u definieert, een kleine hoeveelheid geheugen en geldt er een beperking voor het aantal variabelen dat u kunt definiëren. Als u veel constanten als geheugenvariabelen definieert, kan u dat veel kostbare geheugenruimte kosten.

Constanten weergeven met identificatiesymbolen

De beste manier om constanten weer te geven, is door identificatiesymbolen te definiëren met #define. U gebruikt identificatiesymbolen voor constanten op dezelfde wijze als geheugenvariabelen, met uitzondering van de initialisatie-opdracht.

Het volgende voorbeeld bevat dezelfde code als het vorige, maar hier wordt de instructie #define gebruikt om nMaxKlant te definiëren als een constante. (Zie *Commando's en functies* voor een volledige beschrijving van de syntaxis van #define.)

```
#define nMaxKlant 5000             && nMaxKlant definiëren
DO WHILE nHuidigeKlant < nMaxKlant  && nMaxKlant gebruiken als constante
   :
   nHuidigeKlant = nHuidigeKlant + 1
ENDDO
```

Tijdens het compileren wordt nMaxKlant overal vervangen door 5000. Hierdoor kunt u de doelmatigheid van letterlijke waarden tijdens de uitvoering combineren met de eenduidigheid en het gemak van programmeren met geheugenvariabelen.

#define is vooral handig wanneer u niet-dBASE-instructies aanroept vanuit DLL's waarin onduidelijke integers en hexadecimale getallen worden gebruikt, zoals in de Windows API.

Het *bereik* van een identificatiesymbool is beperkt tot het bestand waarin u het symbool definieert. Dat wil zeggen, een identificatiesymbool wordt alleen vervangen in hetzelfde programmabestand dat de opdracht #define bevat. Als u globale constanten wilt definiëren die u in meerdere programmabestanden kunt gebruiken, moet u ofwel de opdracht #define opnemen in elk bestand, ofwel de opdracht #include gebruiken met een header-bestand. De tweede mogelijkheid is het meest efficiënt. Zie "Header-bestanden invoegen", verderop in dit hoofdstuk, voor meer informatie.

Uitdrukkingen uitbreiden

Met de syntaxis van de instructie #define kunt u parameters opgeven die moeten worden ingevoegd in de vervangende tekst bij het vervangen van de identificatiesymbolen. U kunt tijd besparen door identificatiesymbolen te maken die bij het compileren worden uitgebreid tot veelgebruikte uitdrukkingen.

Hierna volgt een overzicht van de parametersyntaxis voor #define. Zie *Commando's en functies* voor een volledige beschrijving:

```
#define <identificatiesymbool> ( <parameterlijst> ) <vervangende tekst>
```

De naam van het identificatiesymbool

De parameter(s) die moet(en) worden ingevoegd in de vervangende tekst

De tekst, inclusief parameters, die het identificatiesymbool vervangt

Soms moet u een uitdrukking maken waarin verscheidene geneste functie-aanroepen worden gebruikt. Het typen van deze uitdrukkingen en het bijhouden van de haakjes is een vervelend karwei, vooral als u zulke uitdrukkingen herhaaldelijk gebruikt. In de volgende uitdrukking wordt een numerieke waarde geconverteerd naar een reeks, worden de voorloopspaties verwijderd en wordt de tekst gecentreerd:

```
nWaarde = 1200
? CENTER(LTRIM(STR(nWaarde)))
```

Als u het typen van deze uitdrukking wilt vereenvoudigen, kunt u een identificatiesymbool maken waarmee de drie functies worden gecombineerd in één en waarmee de waarde wordt ingevoegd. U hoeft dan alleen de naam van het identificatiesymbool en de parameters (tussen haakjes) te typen in plaats van de hele uitdrukking.

```
#define Center_Waarde(getal) CENTER(LTRIM(STR(getal)))
nWaarde = 12
? Center_Waarde(nWaarde)          && '12' centreren
? Center_Waarde(nWaarde + 20)     && '32' centreren
```

Tijdens het compileren wordt `Center_Waarde(nWaarde)` uitgebreid tot `CENTER(LTRIM(STR(nWaarde)))`.

Parameters doorgeven: identificatiesymbolen versus procedures

Parameters invoegen in identificatiesymbolen lijkt op parameters doorgeven in procedures of functies. De identificatiesymbolen worden vaak *inline-functies*, *pseudofuncties* of *preprocessor-macro's genoemd*. De analogie met procedures heeft echter belangrijke beperkingen. U moet het invoegen van parameters in identificatiesymbolen dan ook niet verwarren met het doorgeven van parameters in procedures.

In het volgende voorbeeld ziet u een identificatiesymbool en een procedure die ogenschijnlijk hetzelfde doen. Let echter op de verschillende resultaten als de doorgegeven parameters een uitdrukking bevatten (in dit geval `nWaarde1+2`).

```
PROCEDURE CenterProc(getal1,getal2)
antwoord = getal1*getal2
RETURN antwoord

#define Center_Waarde(getal1,getal2) CENTER(LTRIM(STR(getal1 * getal2)))
nWaarde1 = 12
nWaarde2 = 24
? CenterProc(nWaarde1,nWaarde2)          && '288'  (12*24) centreren
? Center_Waarde(nWaarde1,nWaarde2)       && '288' centreren
* Vervolgens evalueert de procedure nWaarde1+2 voordat de berekening wordt uitgevoerd
? CenterProc(nWaarde1+2,nWaarde2)        && '360' ((12 + 2) * 24) centreren
* Het identificatiesymbool evalueert nWaarde1+2 echter niet voordat de berekening wordt
* uitgevoerd. De volgorde van berekening (vermenigvuldigen vóór optellen) leidt derhalve
* tot een ander resultaat
? Center_Val(nWaarde1+2,nWaarde2)        && '576' (12+2*24) centreren en evalueren als
                                         && (12+(2*24))
```

Om ervoor te zorgen dat opdrachtelementen op de juiste wijze worden gegroepeerd bij het evalueren van identificatiesymbolen, moet u haakjes toevoegen aan de opdracht #define, zoals u in het volgende voorbeeld ziet.

```
#define Center_Waarde(getal1,getal2) CENTER(LTRIM(STR((getal1)*(getal2))))
```

Voorwaardelijke compilatie

De dBASE-taal heeft altijd *programmaverloop*-commando's bevat, zoals IF...ELSE...ENDIF, waarmee u bepaalt welke commando's tijdens runtime worden uitgevoerd. Op dezelfde manier kunt u in dBASE voor Windows bepalen welke commando's tijdens het compileren worden uitgevoerd met voorwaardelijke instructies, zoals #if...#else...#endif. Deze techniek, die *voorwaardelijke compilatie wordt genoemd*, is vooral handig voor het opsporen van fouten of het ontwerpen van verschillende versies van een applicatie.

De voorwaarden voor het bepalen van de commando's die worden uitgevoerd, zijn gebaseerd op identificatiesymbolen die u definieert met #define. U kunt twee soorten voorwaarden instellen:

- U kunt controleren of een identificatiesymbool *bestaat* met #ifdef (if defined, indien gedefinieerd) of #ifndef (if not defined, indien niet gedefinieerd). Gebruik #ifdef of #ifndef als u identificatiesymbolen wilt controleren die u definieert zonder waarde, zoals `#define Debug`.
- U kunt de *waarde* van een identificatiesymbool controleren met #if. Gebruik #if als u identificatiesymbolen wilt controleren die u definieert met een waarde, zoals `#define Versie 1`.

Bij elke voorwaardelijke instructie kunt u een "else"-optie (#else) gebruiken als u commando's wilt selecteren die moeten worden uitgevoerd als niet aan de voorwaarde wordt voldaan.

In het volgende voorbeeld wordt een voorwaardelijke compilatie ingesteld voor het opsporen van fouten. Als het identificatiesymbool DEBUG is gedefinieerd, wordt de foutopsporingscode gecompileerd. Als het symbool niet is gedefinieerd, wordt `DO MijnProcedure` gecompileerd.

```
* Aanwezigheid van identificatiesymbool Debug controleren.
#ifdef Debug                          && Volgende code compileren als Debug bestaat
   SET ALTERNATE TO DEBUG.TXT
   SET ALTERNATE ON                   && Alternatief uitvoerbestand openen
   LIST STATUS                        && en overzicht van huidige omgeving weergeven
   LIST MEMORY                        && voor dit bestand
   SET ALTERNATE TO
   SET ALTERNATE OFF
#else                                 && Volgende code compileren als Debug niet bestaat
   DO MijnProcedure
#endif
```

U kunt dezelfde resultaten bereiken met het commando IF...ELSE...ENDIF. De voorwaardelijke instructies worden echter tijdens het compileren geëvalueerd, zodat u alleen de code kunt compileren die u wilt starten. Als u alleen de benodigde code compileert, kan dat de prestaties van uw programma ten goede komen. Bovendien is het mogelijk dat code die niet volledig is gecontroleerd op fouten, helemaal niet wordt gecompileerd. Met voorwaardelijke instructies kunt u op eenvoudige wijze onvolledige codesecties markeren en in een later stadium, wanneer u de secties hebt gecontroleerd op fouten, compileren.

U kunt de voorwaardelijke instructies ook gebruiken voor het ontwerpen van meerdere versies van een programma zonder dat u afzonderlijke sets broncode hoeft bij te houden. De techniek lijkt op die uit het vorige voorbeeld, met dien verstande dat u de *waarde* van een identificatiesymbool controleert in plaats van de aanwezigheid ervan.

In het volgende voorbeeld wordt een identificatiesymbool gedefinieerd voor het weergeven van het versienummer van een programma. Hiertoe wordt versie-specifieke code in de opdrachten #if...#endif geplaatst.

```
#define Versie 1    && Versienummer opgeven
#if Versie = 1
   . "Versie 1"     && Code voor versie 1
   :
#else
   ? "Versie 2"     && Code voor versie 2
   :
#endif
```

Als u de code voor Versie 2 wilt compileren, wijzigt u de opdracht #define in `#define Versie 2`.

Voorwaardelijke compilatie voor meerdere versies van dBASE

In dBASE voor Windows vindt u veel nieuwe taalelementen ten opzichte van dBASE IV. Ten behoeve van de compatibiliteit van code uit vorige versies van dBASE, wordt automatisch een identificatiesymbool gemaakt met de naam __dbasewin__ (het woord "dbasewin", voorafgegaan en gevolgd door twee onderstreepstekens). Als u #ifdef gebruikt om uw programma te controleren op de aanwezigheid van __dbasewin__, kunt u alleen de code compileren die bedoeld is voor dBASE voor Windows en kunt u niet-Windows-code compileren met het compileerprogramma van dBASE DOS.

In het volgende voorbeeld wordt de Windows-specifieke code alleen uitgevoerd wanneer deze wordt gecompileerd met dBASE voor Windows. De andere dBASE-code wordt uitgevoerd als deze wordt gecompileerd met het compileerprogramma van dBASE DOS.

```
#ifdef __dbasewin__
   * Windows-specifieke code hier
#else
   * Andere dBASE-code hier
#endif
```

Header-bestanden invoegen

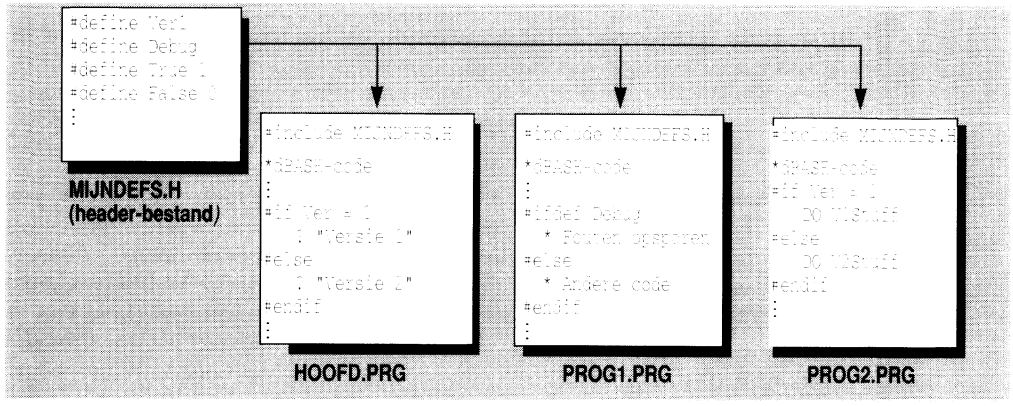
Een *header-bestand* (ook wel een *include-bestand* genoemd) is een programmabestand dat u tijdens het compileren in een ander bestand invoegt met de instructie #include. Meestal voegt u header-bestanden aan het begin (de header) van een programmabestand toe. U kunt header-bestanden echter op elke plaats invoegen. Header-bestanden zijn vooral handig voor het invoegen van globale definities van constanten die u hebt gemaakt met #define.

Tijdens de verwerking van preprocessor-instructies worden regels die beginnen met #include, vervangen door de inhoud van het bestand dat wordt opgegeven in de opdracht #include. Hoewel u in header-bestanden willekeurige dBASE-code kunt opnemen, bevatten deze bestanden doorgaans alleen preprocessor-instructies of definities van klassen.

Identificatiesymbolen die u maakt met #define, zijn bijvoorbeeld alleen van toepassing op het bronbestand met de opdracht #define. Als u een identificatiesymbool in

meerdere programmabestanden gebruikt, moet u de opdracht #define in elk bestand opnemen. Als u dit wilt vermijden, kunt u elke opdracht #define onderbrengen in één header-bestand en kunt u het header-bestand met behulp van #include invoegen in elk bronbestand dat de identificatiesymbolen nodig heeft. In Afbeelding 7.2 wordt dit toegelicht.

Afbeelding 7.2 Header-bestanden invoegen met #include



Als u een van de identificatiesymbolen wilt wijzigen, doet u dat gewoon in het header-bestand. Nadat het programma opnieuw is gecompileerd, wordt de nieuwe definitie gebruikt in alle bronbestanden die de opdracht `#include MIJNDEFS.H` bevatten.

Tabel 7.2 bevat enkele voorbeelden van de elementen die u in header-bestanden kunt opslaan:

Tabel 7.2 Voorbeelden van header-bestanden

Wat u in een header-bestand kunt opslaan	Voorbeeld
Identificatiesymbolen die RGB-kleurwaarden weergeven.	`#define MijnKleur 64,42,155`
Identificatiesymbolen die resultaatwaarden van INKEY(), READKEY(), LASTKEY(), NEXTKEY() weergeven.	`#define CTRL-K 23`
Applicatie-specifieke constanten.	`#define MAXITEMS 10000`
Functieprototypen die u definieert met het commando EXTERN. (Zie Hoofdstuk 25 voor meer informatie.)	`EXTERN CWOPEN MijnFunctie() MijnBib.DLL`
Constanten die worden gebruikt in Windows API-functies.	`#define MB_ICONHAND 0x10L`

Eigen definities van klassen.	`CLASS MijnKlasse OF MYSUPERCLASS`
	`:`
	`ENDCLASS`

Programmeurs gebruiken doorgaans bibliotheken met header-bestanden voor het opslaan van veelgebruikte #define-constanten en -uitdrukkingen. Deze bibliotheken bevinden zich meestal in de subdirectory DBASEWIN\INCLUDE. Raadpleeg de beschrijving van de instructie #include in *Commando's en functies* voor een lijst van de plaatsen waar in dBASE naar header-bestanden wordt gezocht.

Bij dBASE wordt een header-bestand (WINAPI.H) geleverd dat bruikbare definities van constanten en functieprototypen bevat voor het aanroepen van Windows API-functies. Zie Hoofdstuk 25 voor meer informatie.

De dekkingsopties instellen

Met de instructie #pragma COVERAGE kunt u opgeven of een *dekkingsbestand* moet worden gemaakt wanneer uw programma wordt gecompileerd. Een dekkingsbestand is een binair bestand dat cumulatieve informatie bevat over het aantal keren dat elk logisch blok van een programma wordt gestart en afgesloten (en dus volledig uitgevoerd). (Zie Hoofdstuk 3 voor meer informatie over dekkingsbestanden.) Als u `#pragma COVERAGE(ON)` in een programma- of header-bestand gebruikt, hoeft u het commando SET COVERAGE ON niet te gebruiken voordat u het programma compileert. Als u dekkingsanalyse wilt uitschakelen, gebruikt u `#pragma COVERAGE(OFF)`.

Hoofdstuk

8

Fouten verhelpen in programma's

Debuggen is het opsporen van fouten in programma's (*bugs*) en deze vervolgens verwijderen. dBASE voor Windows beschikt over een krachtige debugger waarmee u programmacode kunt weergeven, programma's kunt controleren, afbreekpunten kunt instellen, waarden van variabelen kunt weergeven en de stapel kunt weergeven in vensters die op dat moment zijn geopend. U kunt de programma-uitvoering regel voor regel volgen of u kunt naar bepaalde afbreekpunten gaan om te zien waar programmafouten optreden. Nadat u de code hebt gecorrigeerd, kunt u het programma compileren en opnieuw uitvoeren in de debugger om te zien of het probleem is verholpen.

Over de debugger

De debugger is het belangrijkste hulpmiddel voor het opsporen van fouten in programma's. Als u een programma in de debugger uitvoert, kunt u

- De programma-uitvoering beheren
- De waarden van variabelen, velden, array's, objecten en uitdrukkingen weergeven en wijzigen
- De subroutines weergeven die door het hoofdprogramma worden aangeroepen en zien wanneer deze subroutines worden aangeroepen
- Programma-uitvoering op een willekeurige plaats beëindigen, bijvoorbeeld bij een afbreekpunt dat u hebt ingesteld

Soorten fouten

Programma's kunnen drie soorten fouten bevatten:

- Compilatiefouten (of syntaxisfouten)

- Runtime-fouten
- Logische fouten

Compilatiefouten of syntaxisfouten

Als u een programma compileert, krijgt u van het compileerprogramma bericht over *syntaxisfouten* in het programma. Een syntaxisfout is bijvoorbeeld een opdracht IF zonder een bijbehorende opdracht ENDIF. Een syntaxisfout kan ook een ontbrekend leesteken zijn, bijvoorbeeld een ontbrekende puntkomma aan het einde van een regel die aangeeft dat de volgende regel een vervolg is en geen nieuwe commandoregel. Deze fouten kunt u betrekkelijk gemakkelijk opsporen en verhelpen.

Runtime-fouten

Het is mogelijk dat uw programma wordt gecompileerd en geen syntaxisfouten bevat. Het programma wordt echter niet uitgevoerd omdat het *runtime-fouten* bevat. Deze fouten doen zich voor omdat het programma iets onmogelijks probeert te doen, bijvoorbeeld een niet-bestaande tabel openen. Runtime-fouten zijn gemakkelijk op te sporen, aangezien de programma-uitvoering stopt zodra deze fouten zich voordoen. Als tijdens uitvoering van het programma in dBASE een syntaxisfout of runtime-fout wordt aangetroffen, wordt de programma-uitvoering gestopt en wordt het dialoogvenster **Programmamelding** weergegeven. In dit venster kunt u:

- Programma-uitvoering annuleren
- De fout negeren en doorgaan met programma-uitvoering
- Programma-uitvoering onderbreken
- Het programmabestand openen in de tekst-editor waarbij de cursor zich bevindt op de regel met de fout
- Het programmabestand openen in de debugger waarbij de cursor zich bevindt op de regel met de fout

Logische fouten

Het is mogelijk dat de syntaxis van uw code juist is en dat de code wordt uitgevoerd zonder dat er runtime-fouten optreden, maar dat de code onjuiste resultaten oplevert. Als een procedure bijvoorbeeld moet worden uitgevoerd nadat een actie is opgetreden, is er sprake van een *logische fout* als de actie niet optreedt wanneer u het programma uitvoert.

dBASE IV bevat de volgende commando's waarbij u de schakelopties ON/OFF moet gebruiken als u de debugger wilt starten en andere debug-taken wilt uitvoeren:

- SET STEP
- SET ECHO
- SET TRAP

Deze commando's hebt u in dBASE voor Windows niet meer nodig, want u voert alle debug-taken uit in de debugger. Als het commando SET STEP ON of SET ECHO ON wordt aangetroffen, wordt de debugger geopend. Het commando SET TRAP ON wordt genegeerd. De OFF-instellingen van deze commando's worden eveneens genegeerd.

De debugger starten

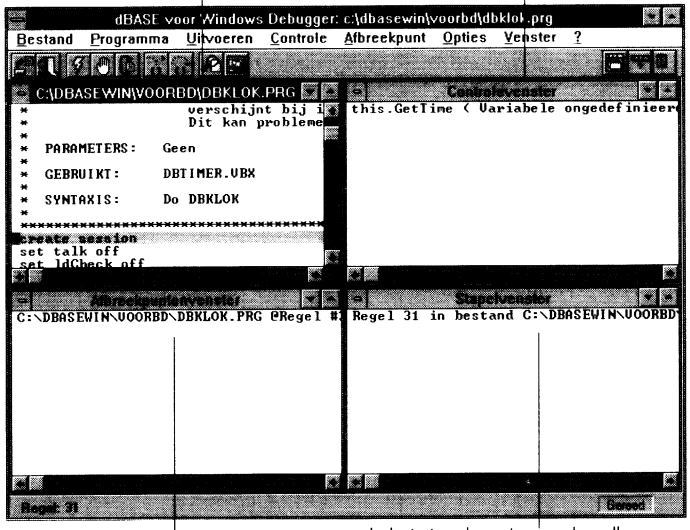
U kunt de debugger starten vanuit dBASE voor Windows of vanuit Programmabeheer van Windows. In het laatste geval wordt de debugger gestart als stand-alone-toepassing.

- Als u de debugger wilt starten vanuit dBASE, gaat u als volgt te werk:
  - Selecteer **Programma's** in het navigatorvenster, rechtsklik op het programma dat u wilt debuggen en kies **Debugger** in het snelmenu
  - Typ het commando DEBUG in het commandovenster of in een programma- of formulierbestand
  - Als het commandovenster invoerfocus heeft, klikt u op **Debugger** op de knoppenbalk
  - Als het commandovenster invoerfocus heeft, kiest u **Programma | Debugger**
- In Programmabeheer van Windows dubbelklikt u op het pictogram **dBASE voor Windows Debugger**.

Afbeelding 8.1 De vensters van de debugger

Het modulevenster bevat de broncode van het programma dat u wilt debuggen

Het controlevenster bevat *controlepunten* voor variabelen, velden, array's, objecten en uitdrukkingen. U stelt deze controlepunten in.



Het afbreekpuntenvenster bevat afbreekpunten (punten waarbij de programma-uitvoering stopt) die u instelt

In het stapelvenster worden alle programma-aanroepen naar andere modulen en procedures bijgehouden. Op de laatste regel staat de huidige subroutine.

Als u de debugger start, is het modulevenster actief. Als u de invoerfocus wilt wijzigen, klikt u op het venster dat u wilt activeren of kiest u het desbetreffende venster in het menu **Venster**. U kunt ook **Naast elkaar** of **Trapsgewijs** kiezen in het menu **Venster** als u de vensters respectievelijk naast elkaar of trapsgewijs wilt weergeven. Als u een of meer vensters tot pictogram verkleint, kunt u de pictogrammen rangschikken met **Venster | Pictogrammen rangschikken**.

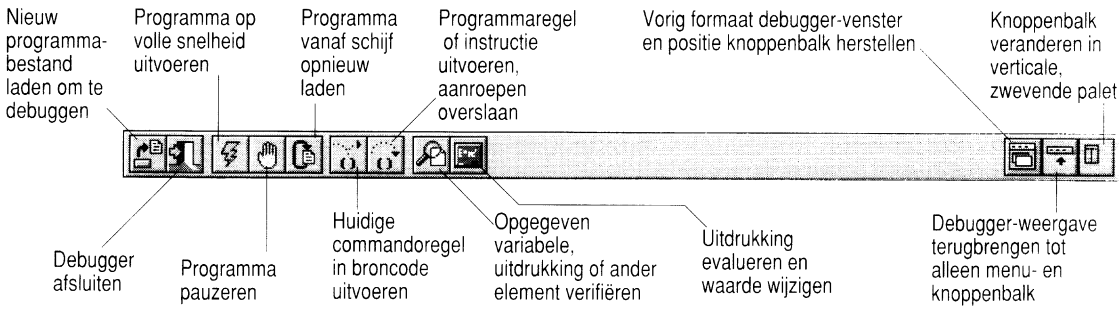
U kunt de positie en de richting van de debugger-knoppenbalk wijzigen en het debugger-applicatievenster vergroten of verkleinen met de drie knoppen aan de rechterkant van de knoppenbalk (Afbeelding 8.2):

- Als u de knoppenbalk onder de debugger-menubalk wilt veranderen in een vrijzwevende verticale palet, klikt u op de knop met de verticale palet uiterst rechts op de knoppenbalk.

- Als u de horizontale knoppenbalk onder de debugger-menubalk weer wilt herstellen, klikt u op de meest linkse van de drie knoppen aan de rechterkant van de knoppenbalk.

- Als u de debugger-weergave wilt terugbrengen tot alleen een menubalk en de knoppenbalk (en u het vensterweergavegebied wilt verbergen), klikt u op de middelste knop van de drie knoppen aan de rechterkant van de knoppenbalk. Klik op de meest linkse knop van deze drie knoppen als u het vorige formaat van de debugger-weergave wilt herstellen.

Afbeelding 8.2 De knoppenbalk van de debugger



Een programmabestand laden

U kunt een programmabestand laden wanneer u de debugger start of nadat u de debugger hebt gestart. Als u een bestand wilt laden wanneer u de debugger start, voert u een van de volgende handelingen uit:

- Selecteer **Programma's** in het navigatorvenster, rechtsklik op het programma dat u wilt debuggen en kies **Debugger** in het snelmenu
- Als het commandovenster actief is, kiest u **Programma | Debugger** en geeft u het programma op dat u wilt debuggen
- In het commandovenster typt u DEBUG <*Bestandsnaam*>, waarbij <*Bestandsnaam*> het programmabestand is dat u wilt debuggen

- Typ DEBUG in het programmabestand als u wilt debuggen
- Als zich een runtime-fout in uw programma voordoet, geeft u op dat u het programma wilt debuggen

Als u een bestand wilt laden nadat u de debugger hebt gestart, kiest u **Bestand | Nieuw** of klikt u op **Programmabestand laden** op de knoppenbalk. Vervolgens geeft u in het dialoogvenster **Applicatie selecteren** het programmabestand op dat u wilt debuggen.

U laadt een formulierbestand (met de extensie .WFM) op dezelfde wijze in de debugger als programmabestanden. U kunt alleen niet rechtsklikken op formulierpictogrammen in het navigatorvenster omdat de opdracht **Debugger** niet in het snelmenu voor formulierbestanden staat.

Als u een programma in de debugger laadt, wordt het programmabestand geopend in het modulevenster waarbij de eerste uitvoerbare programmaregel is gemarkeerd. Op de titelbalk van het debugger-venster wordt de naam van het programmabestand weergegeven. Terwijl u het programma vanuit de debugger uitvoert, schuift de broncode in het venster door naar de volgende uitvoerbare programmaregel. U kunt in de broncode schuiven zonder dat daardoor de programma-uitvoering wordt beïnvloed. Het volgende commando dat wordt uitgevoerd, blijft gemarkeerd, ongeacht de plaats van de cursor in het modulevenster.

Het venster waarin uw programma wordt uitgevoerd, bevindt zich op de achtergrond terwijl het debugger-venster steeds op de voorgrond blijft. U kunt de invoerfocus naar uw programma verplaatsen met *Alt-Tab*. U kunt ook de debugger tot pictogram verkleinen en op het venster met uw programma klikken. Terwijl uw programma actief is, schuift de bijbehorende code nog steeds in het modulevenster. Als u de programma-uitvoering annuleert, blijft de bijbehorende code geopend in het modulevenster.

Subroutines laden

Als u een programma in de debugger hebt geladen, wilt u misschien een of meer van de bijbehorende subroutines laden. Als u een hoofdprogramma uitvoert waarin andere programmabestanden worden aangeroepen, worden de bestanden met subroutines geladen wanneer deze door het hoofdprogramma worden aangeroepen. U kunt echter een subroutine in de debugger laden voordat deze door het programma wordt aangeroepen.

Als u een subroutine wilt laden met behulp van de menu-opdrachten in het debugger-venster, gaat u als volgt te werk:

1 Kies **Bestand | Module laden**

2 Geef in het dialoogvenster **Module selecteren** het subroutine-programmabestand op dat u naast het huidige (hoofd)programmabestand wilt laden

Als u een subroutine wilt laden vanuit het modulevenster, gaat u als volgt te werk:

1 Rechtsklik in het modulevenster zodat het bijbehorende snelmenu verschijnt

2 Kies **Module laden**

3 Geef in het dialoogvenster **Module selecteren** het programmabestand op dat u in de debugger wilt laden

Als u **Bestand | Nieuw** kiest terwijl een programma in de debugger is geladen, wordt u gevraagd of u de huidige debug-sessie wilt beëindigen.

Navigeren in het modulevenster

Met de opdrachten in het menu **Bestand** en het menu **Programma** of met het snelmenu van het modulevenster kunt u de broncode van uw programma onafhankelijk van de programma-uitvoering in de debugger verschuiven in het modulevenster.

Terugkeren naar de tekst-editor

U kunt de programmatekst niet bewerken in het modulevenster. Als u wilt terugkeren naar de tekst-editor om coderegels te bewerken, doet u het volgende:

1 Verplaats de cursor in het modulevenster naar de regel met broncode die u wilt gaan bewerken

2 Kies **Programma | Bewerken**, of kies **Bewerken** in het snelmenu van het modulevenster

Naar een regelnummer gaan

Als u naar een regelnummer in de broncode van het huidige programmabestand wilt gaan, doet u het volgende:

1 Kies **Programma | Ga naar regel** of rechtsklik op het modulevenster en kies **Ga naar regel** in het bijbehorende snelmenu

2 Geef het regelnummer op in het dialoogvenster **Ga naar regel**

Als u in het modulevenster naar een regelnummer gaat, wordt de cursor verplaatst naar het begin van die regel. Dit heeft geen invloed op de programma-uitvoering, die blijft gepauzeerd op de laatste regel waarheen u met behulp van **Overheen stappen** of **Traceren in** bent gegaan, of op het afbreekpunt dat het laatst is uitgevoerd.

De debugger houdt een overzicht bij van alle regelnummers waarheen u tijdens een debug-sessie met **Ga naar regel** gaat. Als u opnieuw naar een van deze regelnummers wilt gaan, doet u het volgende:

1 Kies **Programma | Ga naar regel** of rechtsklik op het modulevenster en kies **Ga naar regel** in het snelmenu

2 Klik op de pijl-omlaag rechts van het vak **Regelnummer** in het dialoogvenster **Ga naar regel**

3 Kies een eerder gebruikt regelnummer in de lijst

De cursor naar de vorige positie verplaatsen

Als u de cursor weer naar de vorige positie wilt verplaatsen, gaat u als volgt te werk:

1 Kies **Programma | Vorige regel** of rechtsklik op het modulevenster

2 Kies **Vorige regel** in het snelmenu van het modulevenster

Tekst zoeken

Als u in het huidige programmabestand naar een tekstreeks wilt zoeken, doet u het volgende:

1 Kies **Programma | Zoeken** of rechtsklik op het modulevenster en kies **Zoeken** in het snelmenu

2 Geef de tekst die u wilt zoeken op in het dialoogvenster **Zoeken**

De zoekbewerking begint op de huidige positie van de cursor. Bij het zoeken wordt geen onderscheid gemaakt tussen hoofdletters en kleine letters. Als de tekst wordt gevonden, verschuift de code in het modulevenster naar de regel met de gezochte tekst en wordt de cursor naar het begin van de tekst verplaatst. Als u nogmaals naar dezelfde tekst wilt zoeken, kiest u **Programma | Volgende zoeken** of kiest u **Volgende zoeken** in het snelmenu van het modulevenster.

De debugger houdt een overzicht bij van alle tekstreeksen waarnaar u tijdens een debug-sessie zoekt. Als u opnieuw naar een van deze tekstreeksen wilt zoeken, gaat u als volgt te werk:

1 Kies **Programma | Zoeken** of rechtsklik op het modulevenster en kies **Zoeken** in het snelmenu

2 Klik op de pijl-omlaag rechts van het vak **Zoekreeks** in het dialoogvenster **Zoeken**

3 Kies een eerder gezochte tekstreeks in de lijst

Naar een procedure gaan

Als u naar een procedure of functie in het huidige programmabestand of een van de bijbehorende procedure- of bibliotheekbestanden wilt gaan, doet u het volgende:

1 Kies **Programma | Ga naar procedure** of rechtsklik op het modulevenster en kies **Ga naar procedure** in het snelmenu

2 Geef de naam van de procedure op in het dialoogvenster **Ga naar procedure**

De zoekbewerking begint op de huidige positie van de cursor in het modulevenster. Als de procedure zich bevindt in het programma dat u debugt, wordt de cursor verplaatst naar de eerste regel van de procedure. Als de procedure zich bevindt in een niet-geopend procedurebestand, wordt de procedure in het modulevenster geladen. Als de procedure niet kan worden gevonden, blijft het huidige programmabestand in het modulevenster staan.

De debugger houdt een overzicht bij van alle procedures waarnaar u tijdens een debug-sessie zoekt. Als u opnieuw naar een van deze programmamodulen wilt gaan, doet u het volgende:

1 Kies **Programma | Ga naar procedure** of rechtsklik op het modulevenster en kies **Ga naar procedure** in het snelmenu

2 Klik op de pijl-omlaag rechts van het vak **Procedurenaam** in het dialoogvenster **Ga naar procedure**

3 Kies een procedurenaam in de lijst

Terugkeren naar de aanroepende procedure (oorsprong)

Als u de cursor naar een procedure hebt verplaatst met **Ga naar procedure**, kunt u terugkeren naar het aanroepende commando. Kies **Programma | Oorsprong** of rechtsklik op het modulevenster en kies **Oorsprong** in het snelmenu.

Programma-uitvoering beheren

U kunt programma-uitvoering in de debugger op de volgende manieren beheren:

Tabel 8.1 Methoden voor het beheer van programma-uitvoering in de debugger

Methode	Type beheer van programma-uitvoering
Animeren	Voert programma's in één keer uit en pauzeert even bij elke regel. Met behulp van **Animatiesnelheid** bepaalt u hoe lang bij elke regel wordt gepauzeerd.
Traceren	Voert programma's regel voor regel uit en pauzeert bij elke regel totdat u opdracht geeft door te gaan met de volgende regel. Geeft de uitvoering van subroutines regel voor regel weer.
Overheen stappen	Voert programma's regel voor regel uit en pauzeert bij elke regel totdat u opdracht geeft door te gaan met de volgende regel. Geeft de uitvoering van subroutines niet regel voor regel weer.
Afbreekpunten instellen	Afbreekpunten zijn bepaalde plaatsen of voorwaarden die u instelt om de uitvoering van een programma te onderbreken en zo het programma op kritieke plaatsen te kunnen controleren. U kunt bijvoorbeeld de waarde van gegevens evalueren of wijzigen, waardoor het programma op die plaats wordt beïnvloed.
Onderbreken	Als u een programma op volle snelheid uitvoert of een programma animeert, kunt u de uitvoering op elk gewenst moment onderbreken.

Animeren

Als u een programma animeert, voert de debugger het programma doorlopend uit. Hierbij wordt bij elke commandoregel gepauzeerd en wordt informatie in alle debugger-vensters tijdens elke pauze bijgewerkt. De taken in uw programma worden uitgevoerd alsof u het programma in dBASE uitvoert. Het verschil is dat u de snelheid van uitvoering kunt bepalen. De broncode verschuift in het modulevenster terwijl de regels worden uitgevoerd. Het woord *Uitvoeren* verschijnt rechts op de statusbalk.

Als u een programma wilt animeren, kiest u **Uitvoeren | Animeren**. Als u de animatiesnelheid wilt instellen, kiest u **Opties | Animatiesnelheid** en geeft u de animatiesnelheid op in het dialoogvenster **Animatiesnelheid instellen** door het schuifblokje op de horizontale schuifbalk te verplaatsen. Als u het schuifblokje naar rechts verplaatst, wordt de animatiesnelheid verhoogd. Als u het schuifblokje naar links verplaatst, wordt de animatiesnelheid verlaagd.

De animatiesnelheid die u instelt, geldt voor alle programma's die u animeert. U kunt de snelheid voor of tijdens programma-animatie wijzigen. Als u **Opties | Animatiesnelheid** kiest terwijl u een programma animeert, kunt u de snelheid van de huidige animatie wijzigen met behulp van het schuifblokje op de schuifbalk.

Traceren

In de debugger kunt u een programma regel voor regel uitvoeren, ofwel een programma *traceren*, en bij elke regel stoppen. Als u in een programma traceert, wordt elke regel in het hoofdprogramma en in alle subroutines die door het hoofdprogramma worden aangeroepen, uitgevoerd en bovendien wordt bij elke regel gepauzeerd. Als u **Overheen stappen** kiest (deze methode wordt in de volgende sectie besproken), wordt het hoofdprogramma uitgevoerd maar worden de regels voor het aanroepen van subroutines overgeslagen.

Als u in een subroutine traceert, wordt de desbetreffende module in het modulevenster geladen en wordt gestopt bij de eerste regel van de subroutine. U kunt doorgaan met traceren in de subroutine en in subroutines op een lager niveau die door de subroutine worden aangeroepen, of u kunt over eventuele aanroepen van subroutines op een lager niveau heen stappen.

Als u wilt traceren in een subroutine, begint u met het hoofdprogramma en traceert u hierin of stapt u over de opdrachten in dit programma heen totdat u bij de opdracht bent die de subroutine aanroept. Vervolgens traceert u in de subroutine als u wilt zien hoe deze regel voor regel in het modulevenster wordt uitgevoerd.

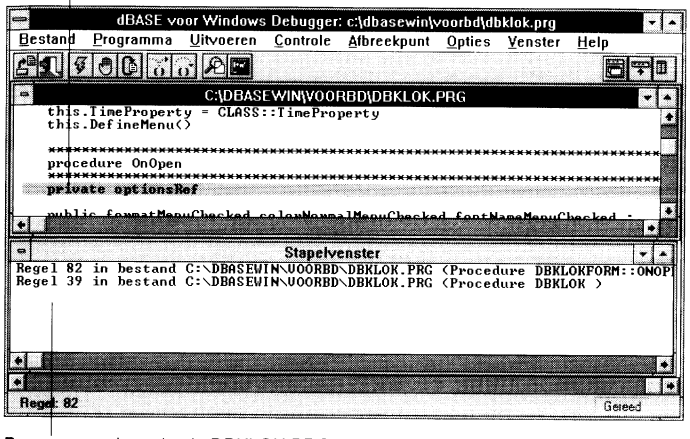
U kunt op een van de volgende manieren een programma en een subroutine traceren:

- Klik op **Traceren in** op de knoppenbalk
- Kies **Uitvoeren | Traceren in**
- Druk op *F7*

Afbeelding 8.3 Traceren in een subroutine

Programma-uitvoering wordt gepauzeerd bij de eerste regel van de procedure OnOpen, een subroutine van het programma DbKlok. Voor OnOpen is dus de methode Traceren gebruikt, niet Overheen stappen. Op dit punt kan worden doorgegaan met traceren of regel voor regel overheen stappen in OnOpen.

Het formaat van het module- en stapelvenster is gewijzigd zodat de inhoud kan worden weergegeven.



Programma-uitvoering in DBKLOK.PRG is gepauzeerd bij regel 82, de eerste regel van de procedure OnOpen. Regel 39 in DBKLOK.PRG is de oorsprong van de programma-uitvoering, de regel die OnOpen heeft aangeroepen.

Overheen stappen

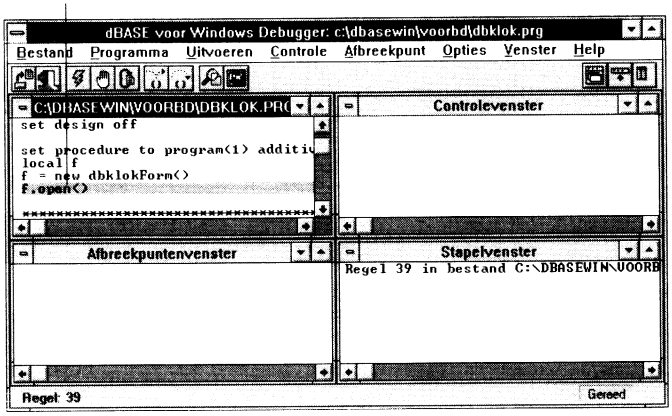
Als u door een programma heen stapt, wordt het hoofdprogramma getraceerd maar worden regels die subroutines aanroepen, overgeslagen. Als u al hebt vastgesteld dat een bepaalde subroutine geen fouten bevat, kunt u over de uitvoering van de subroutine heen stappen en uw aandacht richten op de rest van het programma. De subroutine wordt nog steeds aangeroepen en uitgevoerd. De regel-voor-regel-uitvoering van de subroutine in het modulevenster wordt echter niet weergegeven, waardoor u bij geen enkele regel in de subroutine kunt pauzeren. De debugger stopt bij de eerste commandoregel die volgt op de subroutine.

U kunt op een van de volgende manieren over een programma en over commandoregels die subroutines aanroepen heen stappen:

- Klik op **Overheen stappen** op de knoppenbalk
- Kies **Uitvoeren | Overheen stappen**
- Druk op *F8*

Afbeelding 8.4 Over een aanroep van een subroutine heen stappen

Regel 39 van DBKLOK.PRG, die de procedure DbklokForm aanroept, is de volgende regel die moet worden uitgevoerd. Als u over deze aanroep heen stapt, wordt DbklokForm toch uitgevoerd. De uitvoering wordt echter niet weergegeven in het modulevenster en kan daarom ook niet worden gepauzeerd. De volgende pauze in de programma-uitvoering is bij regel 79.



Afbreekpunten

Een *afbreekpunt* onderbreekt de programma-uitvoering zodat u variabelen, velden, array's, objecten en uitdrukkingen kunt evalueren, de waarde van variabelen, array's en objecten kunt wijzigen en kunt nagaan in welke subroutine het programma zich bevindt.

Als u door een programma heen stapt of in een programma traceert, onderbreekt u het programma in feite bij elke commandoregel. Als u weet dat bepaalde delen van uw programma geen fouten bevatten, hoeft u niet bij elke regel te pauzeren. In plaats daarvan kunt u afbreekpunten instellen op cruciale plaatsen, vervolgens het programma op volle snelheid uitvoeren en de programmawaarden evalueren bij het afbreekpunt. U stelt afbreekpunten in als u een programma in de debugger op volle snelheid animeert of uitvoert.

Als u vermoedt dat een fout op een bepaalde plaats in uw programma optreedt, bijvoorbeeld wanneer een subroutine wordt aangeroepen, kunt u een afbreekpunt op de commandoregel instellen waarmee de subroutine wordt aangeroepen. Vervolgens kunt u vanaf die plaats door het programma stappen of het programma traceren om het probleem op te sporen.

Afbreekpunten kunnen ook worden ingesteld bij voorwaarden in plaats van bij een programma-opdracht. U kunt de programma-uitvoering bijvoorbeeld stoppen als een bepaalde waarde wordt toegewezen aan een variabele. De programma-uitvoering stopt wanneer de waarde voor de eerste keer aan de variabele wordt toegewezen. De uitvoering stopt niet als de waarde nooit aan de variabele wordt toegewezen.

Afbreekpunten instellen

U kunt op een van de volgende manieren een afbreekpunt in uw programma instellen:

- Plaats de cursor in het modulevenster links van de commandoregel die u als afbreekpunt wilt instellen en klik wanneer de cursor verandert in een hand.
- Plaats de cursor in het modulevenster op de commandoregel die u als afbreekpunt wilt instellen en voer een van de volgende handelingen uit:
  - Kies **Afbreekpunt | In-/uitschakelen** of rechtsklik op het afbreekpuntenvenster en kies **In-/uitschakelen** in het snelmenu
  - Druk op *F2*
- Kies **Afbreekpunt | Toevoegen** of rechtsklik op het afbreekpuntenvenster en kies **Toevoegen** in het snelmenu. Geef vervolgens het afbreekpunt op in het dialoogvenster **Afbreekpunt toevoegen**. De typen afbreekpunten die u in dit dialoogvenster kunt opgeven, worden verderop in deze sectie beschreven.

Het afbreekpunt dat u instelt, verschijnt in het afbreekpuntenvenster. Als het afbreekpunt een bepaalde regel is, wordt deze regel ook gemarkeerd in het modulevenster.

Afbreekpunten verwijderen

U kunt afbreekpunten op commandoregels op vrijwel dezelfde manieren verwijderen als u ze instelt:

- Plaats de cursor in het modulevenster links van de commandoregel met het afbreekpunt dat u wilt verwijderen en klik wanneer de cursor verandert in een hand.
- Plaats de cursor in het modulevenster op de commandoregel met het afbreekpunt dat u wilt verwijderen en voer een van de volgende handelingen uit:
  - Kies **Afbreekpunt | In-/uitschakelen** of rechtsklik op het afbreekpuntenvenster en kies **In-/uitschakelen** in het snelmenu
  - Druk op *F2*

Als u een ander type afbreekpunt wilt verwijderen, markeert u het desbetreffende afbreekpunt in het afbreekpuntenvenster en voert u een van de volgende handelingen uit:

- Kies **Afbreekpunt | In-/uitschakelen** of rechtsklik op het afbreekpuntenvenster en kies **In-/uitschakelen** in het snelmenu
- Kies **Afbreekpunt | Verwijderen** of rechtsklik op het afbreekpuntenvenster en kies **Verwijderen** in het snelmenu
- Druk op *F2*

Als u alle huidige afbreekpunten wilt verwijderen, kiest u **Afbreekpunt | Alle verwijderen** of rechtsklikt u op het afbreekpuntenvenster en kiest u **Alle verwijderen** in het snelmenu.

Afbreekpunten bewerken

U kunt op een van de volgende manieren afbreekpunten selecteren die u wilt bewerken:

- Dubbelklik op het afbreekpunt in het afbreekpuntenvenster

- Selecteer het afbreekpunt in het afbreekpuntenvenster en kies **Afbreekpunt | Bewerken** of rechtsklik op het afbreekpuntenvenster en kies **Bewerken** in het snelmenu

Vervolgens kunt u de beschrijving van het afbreekpunt bewerken in het dialoogvenster **Afbreekpunt bewerken**.

Programma-specifieke of algemene afbreekpunten instellen

U kunt een afbreekpunt maken voor één programmabestand of u kunt een algemeen afbreekpunt maken voor alle programmabestanden in de huidige debugger-sessie. Als u een programmabestand opgeeft, kunt u een van de volgende elementen instellen als een programma-specifiek afbreekpunt:

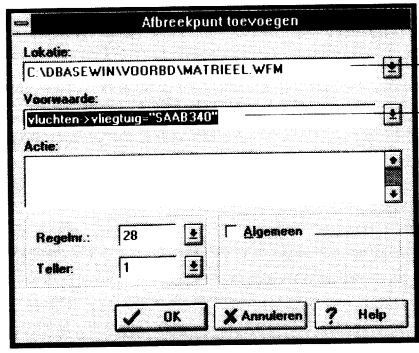
- Een voorwaarde indien deze wordt geëvalueerd als True (waar)
- Het nummer van een programmaregel

Als u alle programmabestanden opgeeft, kunt u een van de volgende elementen instellen als een *algemeen* afbreekpunt:

- Een voorwaarde indien deze wordt geëvalueerd als True
- Een uitdrukking indien de waarde ervan verandert

Wanneer u het dialoogvenster **Afbreekpunt toevoegen** voor de eerste keer opent, verschijnt de naam van het huidige programmabestand in het vak **Lokatie** en is het aankruisvakje **Algemeen** niet ingeschakeld. Als u het dialoogvenster **Afbreekpunt bewerken** opent om een bestaand afbreekpunt in het huidige programmabestand te bewerken, verschijnt de naam van dat programmabestand eveneens in het vak **Lokatie** en is het aankruisvakje **Algemeen** evenmin ingeschakeld. Logische voorwaarden, regelnummers of tellers die u opgeeft, zijn van toepassing op het programmabestand in het vak **Lokatie**.

Afbeelding 8.5 Een programma-specifiek afbreekpunt



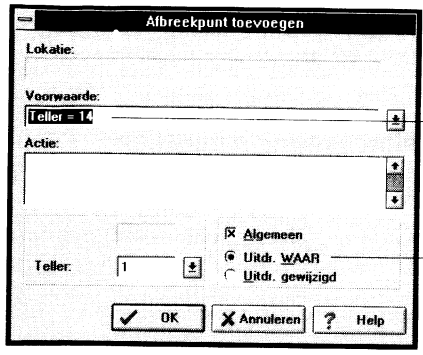
Het huidige programma dat u debugt, verschijnt in het vak Lokatie (standaardinstelling)

Dit afbreekpunt geldt alleen voor MATRIEEL.WFM. Als de voorwaarde wordt geëvalueerd als True, wordt de programma-uitvoering onderbroken.

Zolang het aankruisvakje Algemeen niet is ingeschakeld, is het afbreekpunt dat u opgeeft alleen van toepassing op het programma in het vak Lokatie

Als u het aankruisvakje **Algemeen** inschakelt, wordt het vak **Lokatie** alleen-lezen, verdwijnt het vak **Regelnr** en worden de keuzerondjes **Uitdr. WAAR** en **Uitdr. gewijzigd** weergegeven. Het keuzerondje **Uitdr. WAAR** is standaard geselecteerd (Afbeelding 8.6). Als u het keuzerondje **Uitdr. gewijzigd** selecteert, verandert het vak **Voorwaarde** in het vak **Uitdrukking** (Afbeelding 8.7). Een uitdrukking die u typt in het vak **Voorwaarde** of **Uitdrukking**, is van toepassing op alle programmabestanden die u tijdens de huidige debug-sessie uitvoert.

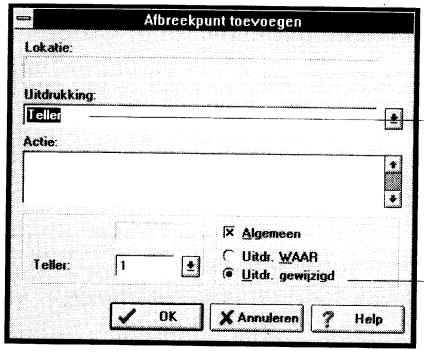
Afbeelding 8.6 Een algemeen afbreekpunt voor voorwaarde ingesteld op True



Dit afbreekpunt geldt voor alle programmabestanden die u in de huidige debug-sessie controleert

Als u Uitdr. WAAR selecteert, moet u een True- of False-uitdrukking in het vak Voorwaarde opgeven. Als de uitdrukking wordt geëvalueerd als True in een programma dat u debugt, wordt de programma-uitvoering onderbroken.

Afbeelding 8.7 Een algemeen afbreekpunt voor gewijzigde uitdrukkingt



Dit afbreekpunt geldt voor alle programmabestanden die u in de huidige debug-sessie controleert

Als u Uitdr. gewijzigd selecteert, hoeft de uitdrukking die u opgeeft in het vak Uitdrukking niet te worden geëvalueerd als True of False. Als de waarde van de uitdrukking wordt gewijzigd in een programma dat u debugt, wordt de programma-uitvoering onderbroken.

Afbreekpunt instellen voor een uitdrukking

In de dialoogvensters **Afbreekpunt toevoegen** en **Afbreekpunt bewerken** kunt u opgeven dat de programma-uitvoering moet worden onderbroken als een uitdrukking wordt geëvalueerd als True. U kunt een voorwaarde opgeven voor één programmabestand of voor alle programmabestanden. Als u een afbreekpunt wilt instellen voor een bepaald programmabestand, typt u de uitdrukking in het vak **Voorwaarde**.

Als u het aankruisvakje **Algemeen** inschakelt om een afbreekpunt in te stellen voor alle programmabestanden, kunt u een uitdrukking opgeven als afbreekpunt. Selecteer het keuzerondje **Uitdr. WAAR** als u een uitdrukking wilt opgeven waarmee de programma-uitvoering wordt onderbroken als de uitdrukking wordt geëvalueerd als True. Typ de uitdrukking in het vak **Voorwaarde** (Afbeelding 8.6). Selecteer het keuzerondje **Uitdr. gewijzigd** als u een uitdrukking wilt opgeven waarmee de programma-uitvoering wordt onderbroken als de waarde van de uitdrukking wordt gewijzigd. Typ deze uitdrukking in het vak **Uitdrukking** (Afbeelding 8.7).

Afbreekpunt instellen voor een bepaald regelnummer

Als u een afbreekpunt instelt voor een bepaald programmabestand, kunt u een commandoregel als afbreekpunt opgeven in het dialoogvenster **Afbreekpunt toevoegen** of

Afbreekpunt bewerken. Typ het regelnummer in het vak **Regelnr**. Als u al een afbreekpunt op een bepaalde commandoregel hebt ingesteld en u bewerkt vervolgens het afbreekpunt, verschijnt het regelnummer in het vak **Regelnr** van het dialoogvenster **Afbreekpunt bewerken**. Als u vervolgens het regelnummer wijzigt, wordt de oude opdrachtregel door de nieuwe vervangen als afbreekpunt.

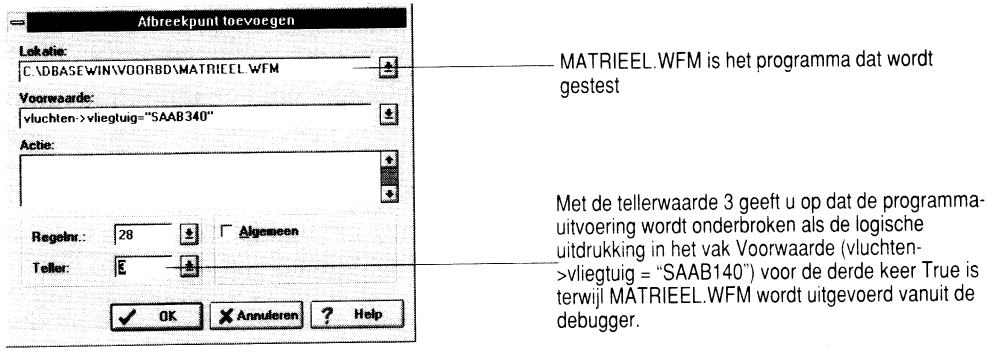
Een teller opgeven

Zodra u een specifiek of algemeen afbreekpunt hebt ingesteld, kunt u opgeven dat de programma-uitvoering pas wordt onderbroken wanneer een aantal keren is voldaan aan de voorwaarde voor het afbreekpunt. Met behulp van een *teller* geeft u het aantal keren op dat het afbreekpunt moet optreden voordat de programma-uitvoering wordt onderbroken.

Als u een afbreekpunt hebt ingesteld in een commandoconstructie, zoals een DO WHILE-lus, kunt u de programma-uitvoering bijvoorbeeld onderbreken wanneer de lus voor de derde keer wordt uitgevoerd door het programma. Als de lus helemaal niet of maar twee keer wordt uitgevoerd, wordt het programma doorlopend uitgevoerd en wordt niet gepauzeerd bij het afbreekpunt. Dit geeft mogelijk een fout in het programma aan.

Als u de programma-uitvoering wilt onderbreken als aan de voorwaarde voor een afbreekpunt een aantal keren is voldaan, geeft u het aantal keren op in het vak **Teller** in het dialoogvenster **Afbreekpunt toevoegen** of **Afbreekpunt bewerken** (Afbeelding 8.8).

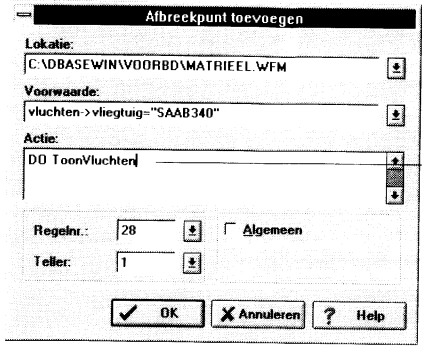
Afbeelding 8.8 Een teller opgeven voor een afbreekpunt



Een actie uitvoeren wanneer zich een afbreekpunt voordoet

Met een programma-specifiek of een algemeen afbreekpunt kunt u een actie opgeven die moet worden uitgevoerd wanneer het afbreekpunt zich voordoet. De actie is bijvoorbeeld een uitdrukking waarmee een variabele op de beginwaarde wordt ingesteld, of een commando DO waarmee een procedure wordt uitgevoerd. Typ de actie, in dBASE-code, in het vak **Actie** in het dialoogvenster **Afbreekpunt toevoegen** of **Afbreekpunt bewerken** (zie Afbeelding 8.9).

Afbeelding 8.9 Een actie voor een afbreekpunt opgeven



Als de voorwaarde voor het afbreekpunt (vluchten->vliegtuig = "SAAB140") True is, wordt programma-uitvoering (van MATRIEEL.WFM) onderbroken en wordt de actie DO ToonVluchten uitgevoerd

Een programma op volle snelheid uitvoeren vanuit de debugger

Als u een programma animeert, er doorheen stapt of traceert, voert u het programma uit. Met animeren, overheen stappen en traceren beheert u echter de uitvoering van het programma, zodat u de resultaten regel voor regel kunt weergeven. Bij animeren kunt u de tijd instellen die moet worden gepauzeerd voordat elke regel wordt uitgevoerd. Bij overheen stappen en traceren kunt u bepalen wanneer de volgende regel wordt uitgevoerd.

Als u afbreekpunten in uw programma hebt ingesteld, kunt u het programma animeren of op volle snelheid uitvoeren. Op deze manier wordt het programma op de normale wijze uitgevoerd tot aan het eerste afbreekpunt. Vervolgens kunt u besluiten dat u vanaf het afbreekpunt door wilt gaan met de uitvoering van het programma of dat u vanaf het afbreekpunt het programma wilt traceren, animeren of eroverheen stappen. Als u een programma uitvoert vanuit de debugger, kunt u de gedeelten die goed werken op volle snelheid uitvoeren, zodat u zich kunt concentreren op problematische gedeelten.

Als u een programma vanuit de debugger op volle snelheid wilt uitvoeren, zonder dat bij elke regel wordt gepauzeerd, voert u een van de volgende handelingen uit:

- Klik op **Uitvoeren** op de knoppenbalk
- Kies **Uitvoeren | Uitvoeren**
- Druk op *F9*

Uitvoeren tot aan de cursor

Als u een programma wilt uitvoeren tot aan de plaats van de cursor in het modulevenster, kiest u **Uitvoeren | Ga naar cursor**. In feite stelt u hiermee een afbreekpunt in op de regel met de cursor.

Uitvoeren tot aan een opdracht RETURN

Als u een programma wilt uitvoeren tot aan het punt dat het programma terugkeert van een subroutine naar de aanroepende routine, kiest u **Uitvoeren | Tot Return**. In feite stelt u hiermee een afbreekpunt in bij de eerste opdracht RETURN die door dBASE wordt aangetroffen.

Parameters voor het programma opgeven

Als u in uw programma argumenten, of *parameters*, kunt gebruiken, kunt u die opgeven voordat u het programma uitvoert vanuit de debugger. Kies **Uitvoeren | Parameterargumenten** en geef de parameterargumenten op in het dialoogvenster **Parameterargumenten**.

Programma-uitvoering onderbreken

Naast overheen stappen, traceren en afbreekpunten instellen, kunt u een programma op elk gewenst moment onderbreken terwijl u het animeert of op volle snelheid uitvoert. Hiervoor voert u een van de volgende handelingen uit:

- Klik op de knop **Onderbreken** op de knoppenbalk
- Kies **Uitvoeren | Onderbreken**

Een programma onderbreken is niet hetzelfde als een programma verwijderen uit het geheugen. Als u een programma tijdens de uitvoering onderbreekt, is dat hetzelfde als wanneer u het programma pauzeert tijdens overheen stappen of traceren. U onderbreekt het programma bij de huidige regel, die gemarkeerd blijft in het modulevenster. Omdat het programma nog in het geheugen is geladen en enkel is onderbroken, wordt het technisch gesproken nog steeds uitgevoerd. Het woord Uitvoeren wordt nog steeds rechts op de statusbalk weergegeven.

Programma's op beginwaarden instellen

Als u een programma vanuit de debugger op de beginwaarden instelt, gebeurt het volgende:

1. Het programma wordt onderbroken als het op dat moment wordt uitgevoerd
2. Het programma wordt uit het geheugen verwijderd
3. Het programma wordt opnieuw in het geheugen en in de debugger geladen, en vervolgens uitgevoerd vanaf de eerste regel

Als u een programma dat wordt uitgevoerd, op de beginwaarden wilt instellen, voert u een van de volgende instructies uit:

- Klik op de knop **Beginwaarden** op de knoppenbalk
- Kies **Uitvoeren | Beginwaarden**

Programma's beëindigen en uit het geheugen verwijderen

Als u **Uitvoeren | Beëindigen** kiest, wordt het programma uit het geheugen en uit de debugger verwijderd. De debugger blijft actief, zodat u een ander programma kunt laden dat u wilt debuggen.

Actie-afhandelingroutines debuggen

Als u actie-afhandelingsroutines wilt debuggen, gebruikt u dezelfde methoden (animeren, overheen stappen, traceren, afbreekpunten instellen, enzovoort) als voor het debuggen van andere code.

De algemene stappen zijn als volgt:

1 Laad het programmabestand met de actie-afhandelingsroutine in de debugger.

2 Stel een afbrekingspunt in in de actie-afhandelingsroutinecode.

3 Voer het programma uit op volle snelheid of stap er doorheen of traceer het.

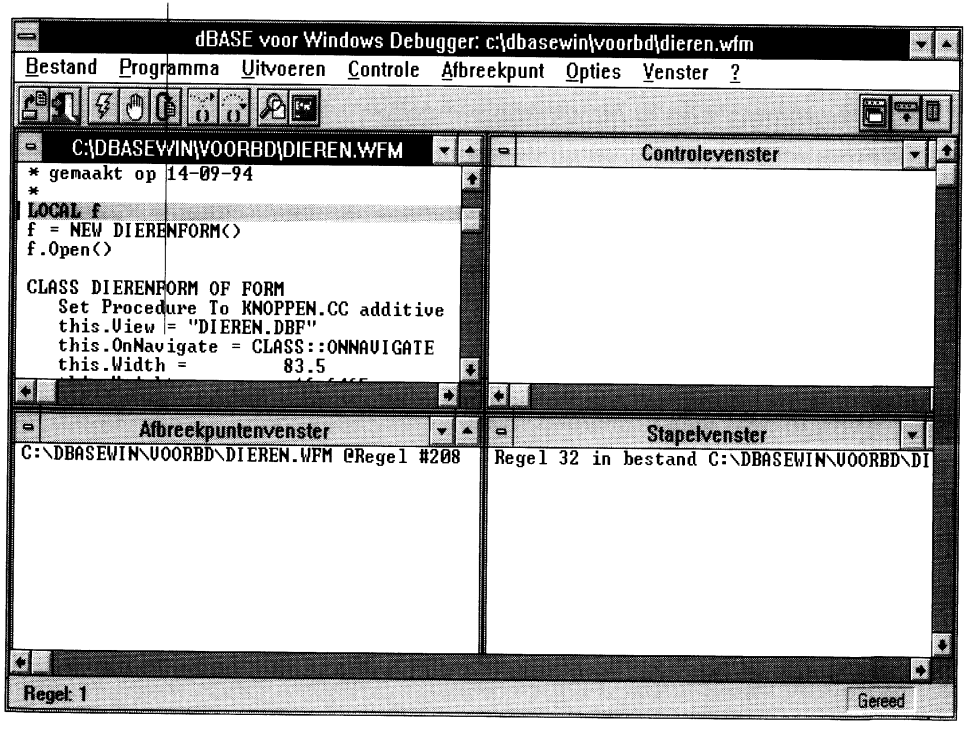
   Als het programma een formulier opent, ontvangt dit de focus.

4 Voer in het formulier de handelingen uit waarmee de actie wordt gestart die behoort bij de actie-afhandelingsroutine. Als de actie-afhandelingsroutine bijvoorbeeld een OnClick-procedure voor een knop is, klikt u op de knop om de OnClick-actie te starten.

Als de actie-afhandelingsroutine de regel of voorwaarde bereikt die wordt aangegeven door het onderbrekingspunt, ontvangt de debugger opnieuw de focus. U kunt de code dan verifiëren of traceren of er doorheen stappen of andere debug-taken uitvoeren.

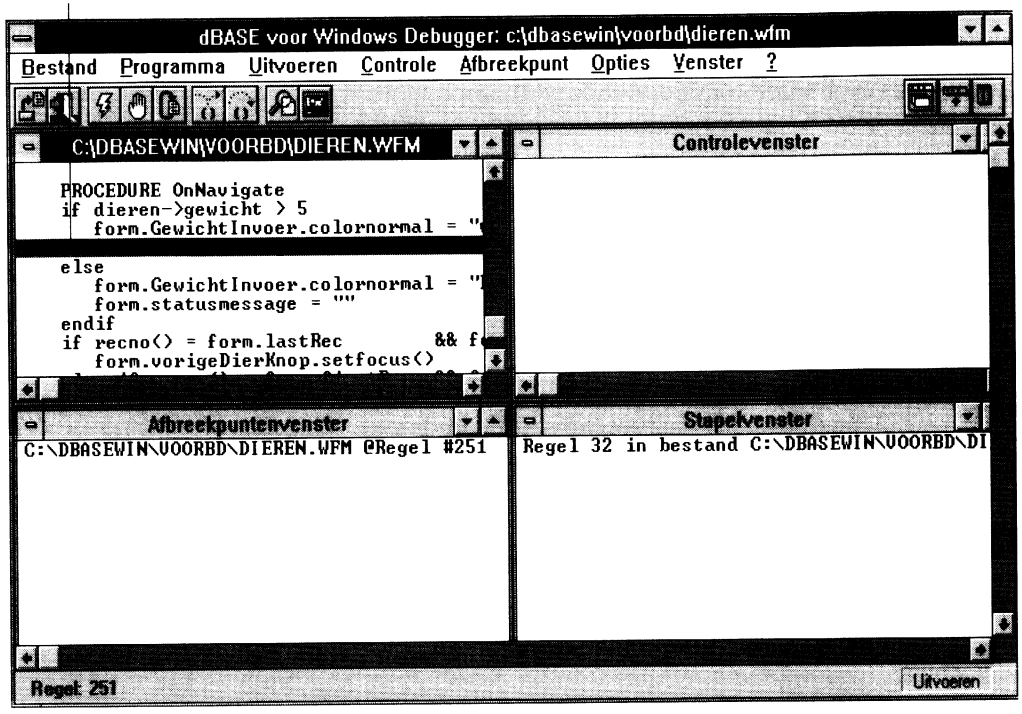
Afbeelding 8.10 Een actie-afhandelingsroutine OnNavigate

In het formulier Dieren is op deze regel een actie-afhandelingsroutine gedefinieerd met de naam ONNAVIGATE die is gekoppeld aan de actie OnNavigate.



Afbeelding 8.11 Een afbreekpunt in een actie-afhandelingsroutine

In deze regel ziet u een afbreekpunt dat voor de procedure ONNAVIGATE is ingesteld binnen de opdracht IF Dieren->gewicht > 5. Als u het formulier start en naar een ander record gaat, wordt de uitvoering gestopt en ontvangt de debugger de focus als de waarde voor het veld GEWICHT groter is dan 5.



De stapel weergeven

Als u een programma uitvoert in de debugger, worden alle aanroepen van andere programma's, procedures of functies bijgehouden in het stapelvenster. Elke regel in het stapelvenster geeft de bestandsnaam en het regelnummer weer van subroutine-aanroepen en aanroepende routines in de volgorde waarin deze zich voordoen (zie Afbeelding 8.3, "Traceren in een subroutine"). In het stapelvenster ziet u dus een weergave van het verloop van uw programma van hoofdprogramma naar subroutine, weer terug naar het hoofdprogramma, of naar een subroutine op een ander niveau, enzovoort.

Telkens wanneer de programma-uitvoering wordt onderbroken, bijvoorbeeld bij elke regel (als u overheen stapt, traceert en animeert) of bij elk afbreekpunt, wordt het stapelvenster bijgewerkt.

U kunt de invoerfocus van het modulevenster verplaatsen naar de lokatie van een subroutine-aanroep:

1 Selecteer in het stapelvenster de regel van de subroutine die u wilt weergeven in het modulevenster

2 Rechtsklik op het stapelvenster en kies **Ga naar bronregel** in het snelmenu

Als de weergave van de broncode in het modulevenster verschuift naar de commandoregel die de subroutine aanroept, wordt de programma-uitvoering op dezelfde plaats onderbroken. Als u vervolgens doorgaat met de programma-uitvoering, wordt de uitvoering hervat vanaf de plaats van de onderbreking en verschuift de weergave in het modulevenster terug naar de volgende commandoregel die wordt uitgevoerd.

Uitdrukkingen controleren

Als u een programma uitvoert vanuit de debugger, kunt u uitdrukkingen of de waarden van uitdrukkingen, zoals variabelen, velden, array's en objecten, controleren. In de debugger kunt u elke uitdrukking controleren in het controlevenster. Dit geldt ook voor uitdrukkingen die niet in het programma voorkomen. Bovendien kunt u een *controlepunt* instellen voor een uitdrukking voordat de variabelen, velden, array's of objecten in de uitdrukking door het programma worden geïnitialiseerd.

Telkens wanneer de programma-uitvoering wordt gepauzeerd, worden controlepunten geëvalueerd en worden de huidige waarden ervan tussen haakjes weergegeven in het controlevenster. U ziet de waarde van een controlepunt terwijl deze waarde op een willekeurige plaats in uw programma wordt gewijzigd. U kunt op deze manier zien of bijvoorbeeld een onjuiste waarde wordt toegewezen aan een variabele of de juiste waarde op een verkeerde plaats in het programma.

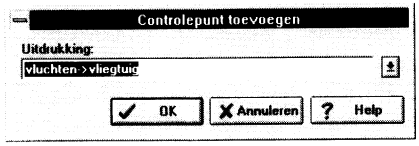
Controlepunten toevoegen

Als u een controlepunt wilt toevoegen, kunt u dat op drie manieren doen:

- Markeer een uitdrukking, zoals een variabele, of verplaats de cursor in het modulevenster naar de uitdrukking in de broncode en voer een van de volgende handelingen uit:
  - Kies **Controle | Toevoegen** of rechtsklik op het modulevenster en kies **Controleren** in het snelmenu
  - Druk op *Ctrl-W*

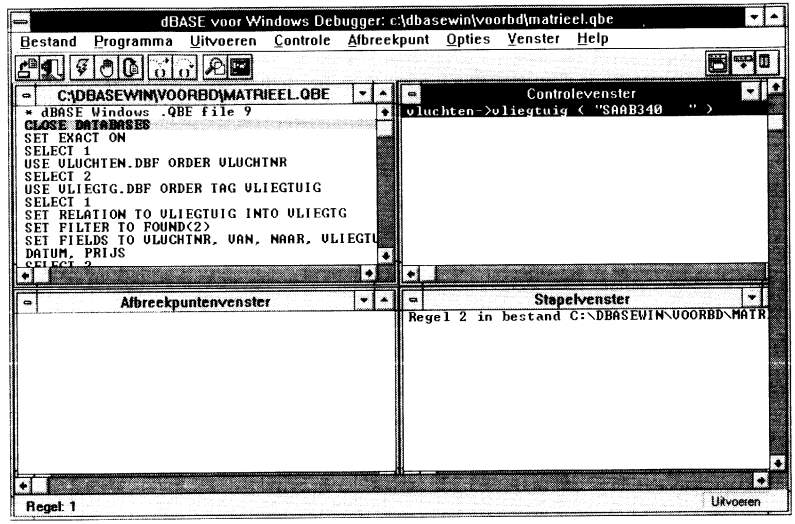
  De geselecteerde uitdrukking verschijnt in het dialoogvenster **Controlepunt toevoegen**.
- Kies **Controle | Toevoegen** of rechtsklik op het controlevenster en kies **Toevoegen** in het snelmenu. Typ vervolgens de uitdrukking die u wilt controleren in het dialoogvenster **Controlepunt toevoegen**.
- Druk op *Ctrl-W* en typ de uitdrukking die u wilt controleren in het dialoogvenster **Controlepunt toevoegen**.

Afbeelding 8.12 Het dialoogvenster Controlepunt toevoegen



Nadat u **OK** hebt gekozen in het dialoogvenster **Controlepunt toevoegen**, verschijnt de opgegeven uitdrukking in het controlevenster. Als de uitvoering van het programma nog niet voorbij het punt is waar de variabelen, velden, array's of objecten in de uitdrukking worden geïnitialiseerd, zijn de haakjes rechts van de uitdrukking in het controlevenster leeg.

Afbeelding 8.13 Een uitdrukking controleren



Controlepunten bewerken

Als u een bestaand controlepunt wilt bewerken, doet u het volgende:

1 Selecteer het controlepunt in het controlevenster

2 Kies **Controle | Bewerken** of rechtsklik op het controlevenster en kies **Bewerken** in het snelmenu

3 Bewerk de uitdrukking in het dialoogvenster **Controlepunt bewerken**

In het dialoogvenster **Controlepunt bewerken** klikt u op de pijl-omlaag rechts van het vak **Uitdrukking** als u een lijst met de huidige controlepunten wilt weergeven. U kunt een van deze controlepunten selecteren en vervolgens bewerken in het dialoogvenster **Controlepunt bewerken**. Alleen het laatste controlepunt dat u bewerkt in de huidige sessie van het dialoogvenster wordt bijgewerkt als u **OK** kiest.

De waarde van controlepunten wijzigen

U kunt de waarde van een uitdrukking niet alleen controleren, maar ook wijzigen. U wijzigt de waarde van een uitdrukking als u het verloop van een programma wilt wijzigen. Als in uw programma bijvoorbeeld niet op de juiste plaasts de juiste waarde aan een variabele wordt toegewezen, kunt u een controlepunt toevoegen voor die variabele, kunt u de programma-uitvoering op die plaats onderbreken (bijvoorbeeld door een afbreekpunt in te stellen op de regel waar de juiste waarde aan de variabele moet worden toegewezen) en kunt u de juiste waarde zelf aan de variabele toewijzen in het controlevenster.

Als u de waarde van een uitdrukking wilt wijzigen die u op dat moment controleert in het controlevenster, gaat u als volgt te werk:

1 Selecteer het controlepunt in het controlevenster

2 Kies **Controle | Wijzigen** of rechtsklik op het controlevenster en kies **Wijzigen** in het snelmenu

3 Typ de gewenste waarde voor de uitdrukking in het dialoogvenster dat verschijnt

Als u **OK** kiest, verschijnt de waarde die u hebt getypt tussen haakjes naast het controlepunt in het controlevenster. Vervolgens kunt u het programma uitvoeren om te zien of het programma met de juiste waarde op de juiste plaats correct werkt.

Uitdrukkingen verifiëren

Een uitdrukking verifiëren is niet hetzelfde als een uitdrukking controleren. Een controlepunt kan te allen tijde worden ingesteld, maar een verificatie alleen *nadat* de variabelen, velden, array's of objecten in de uitdrukking zijn geïnitialiseerd. Met andere woorden, de uitdrukking moet betekenis hebben in de huidige context van het programma. Een controle geeft alleen de waarde van een array of object weer, terwijl de verificatie van een array of object de waarden van alle bijbehorende elementen of kenmerken oplevert.

Verificaties instellen

Als u een uitdrukking wilt verifiëren, kunt u dat op een van de volgende manieren doen:

- Markeer de uitdrukking of verplaats de cursor in het modulevenster naar de uitdrukking in de broncode en doe voer een van de volgende handelingen uit:

  - Klik op **Verifiëren** op de knoppenbalk
  - Kies **Programma | Verifiëren** of rechtsklik op het modulevenster en kies **Verifiëren** in het snelmenu
  - Druk op *Ctrl-V*

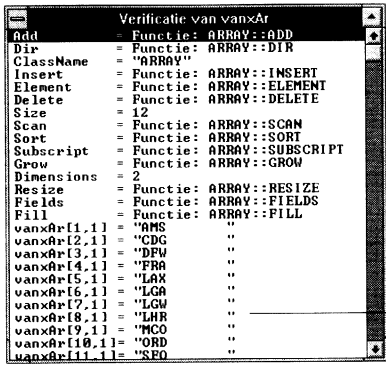
  De geselecteerde uitdrukking verschijnt in het dialoogvenster **Verifiëren**.

- Klik op de knop **Verifiëren** op de knoppenbalk en typ de uitdrukking die u wilt verifiëren in het dialoogvenster **Verifiëren**.

- Kies **Programma | Verifiëren** of rechtsklik op het modulevenster en kies **Verifiëren** in het snelmenu. Typ vervolgens de uitdrukking die u wilt verifiëren in het dialoogvenster **Verifiëren**.
- Druk op *Ctrl-V* en typ de uitdrukking die u wilt verifiëren in het dialoogvenster **Verifiëren**.

Als u op OK klikt, verschijnt het verificatievenster met de opgegeven uitdrukking en bijbehorende waarde. Als u een array of een object hebt opgegeven, worden alle waarden van array-elementen of objectkenmerken eveneens in het verificatievenster weergegeven.

Afbeelding 8.14 Het verificatievenster



Het verificatievenster voor het array vanxAr bevat de waarden van de array-elementen

VanxAr, een array, bevat veel meer elementen. U kunt door het venster schuiven om alle kenmerken weer te geven.

Het verificatievenster

Elk verificatievenster heeft een bijbehorend snelmenu, dat u kunt openen door te rechtsklikken op het venster. Als u een element in een verificatievenster selecteert en vervolgens het desbetreffende snelmenu opent, beschikt u over de volgende keuzemogelijkheden:

- **Bereik**. Hiermee stelt u een bereik (een waarde voor een boven- en ondergrens) in voor het geselecteerde element.
- **Wijzigen**. Hiermee wijzigt u de waarde van het geselecteerde element.
- **Aflopend**. Hiermee stelt u een verificatie in (opent u een afzonderlijk verificatievenster) voor het geselecteerde element in het huidige verificatievenster.
- **Hex-expansie**. Hiermee geeft u de hexadecimale waarde weer van het geselecteerde element als dat element een binaire reeksvariabele is.
- **Nieuwe uitdrukking verifiëren**. Hiermee verifieert u een andere uitdrukking.

In de volgende secties worden deze keuzen in het snelmenu van het verificatievenster beschreven.

Een waardenbereik instellen

Als u het waardenbereik wilt beperken voor de variabelen, velden, array's of objecten die u verifieert, doet u het volgende:

1 Selecteer het element in het verificatievenster

2 Rechtsklik in het verificatievenster en kies **Bereik** in het snelmenu van dit venster

3 Geef in het dialoogvenster **Bereik instellen** een onder- en bovengrens op in de desbetreffende vakken

Voor het testen van ORDERS.WFM kunt u bijvoorbeeld alleen klantnummers tussen 2000 en 2199 toestaan voor het veld Orders->Klntnr in het invoervak KLANTNRINVOER.

De waarde wijzigen van een element dat u verifieert

Als u de waarde wilt wijzigen van de variabelen, array's, velden of objecten die u verifieert, gaat u als volgt te werk:

1 Selecteer het element in het verificatievenster

2 Rechtsklik op het verificatievenster en kies **Wijzigen** in het snelmenu

3 Vervang de huidige waarde van het element door de gewenste waarde in het dialoogvenster **Wijziging van "waarde"**

U kunt bijvoorbeeld MATRIEEL.WFM testen waarbij de waarde {14-2-95} wordt toegewezen aan het veld Vluchten->Datum, dat wordt weergegeven door de variabele Datum in het formulier VLCHTINF.WFM. Dit formulier is een subroutine van MATRIEEL.WFM.

Objectkenmerken of array-elementen verifiëren

U kunt het kenmerk van een object of het element van een array weergeven in een afzonderlijk verificatievenster. Met andere woorden, u kunt een afzonderlijke verificatie instellen voor het objectkenmerk of array-element. Hiertoe gaat u als volgt te werk:

1 Selecteer het objectkenmerk of array-element in het verificatievenster

2 Rechtsklik op het verificatievenster en kies **Aflopend** in het snelmenu

Er wordt een nieuw verificatievenster geopend met alleen het objectkenmerk of array-element plus de bijbehorende waarde.

Hexadecimale waarden van reeksen weergeven

Als u een reeksvariabele verifieert, kunt u de hexadecimale waarde van de huidige waarde van de variabele weergeven. Hiertoe gaat u als volgt te werk:

1 Selecteer de variabele in het verificatievenster

2 Rechtsklik op het verificatievenster en kies **Hex-expansie** in het snelmenu

De hexadecimale waarde van de huidige waarde van de variabele verschijnt in het dialoogvenster **Hexadecimaal**.

Een nieuwe uitdrukking verifiëren

Als u in een verificatievenster werkt en u wilt een verificatie instellen voor een andere uitdrukking, kunt u rechtsklikken op het verificatievenster en kunt u vervolgens **Nieuwe uitdrukking verifiëren** kiezen in het snelmenu. U hoeft dus niet eerst **Programma | Verifiëren** te kiezen of **Verifiëren** in het snelmenu van het modulevenster.

Controleren, verifiëren en bereik

De debugger houdt rekening met het bereik van variabelen waarvoor u controlepunten en verificaties instelt. Stel dat u een controlepunt en een verificatie instelt voor een geheugenvariabele van het type LOCAL. Als de programma-uitvoering naar een subroutine gaat die niet behoort tot het bereik van de variabele, wordt de waarde van het controlepunt "ongedefinieerd" en wordt niets weergegeven in het verificatievenster voor de variabele. Als echter aan een andere variabele met dezelfde naam een waarde in de subroutine wordt toegewezen die buiten het bereik van de oorspronkelijke LOCAL-variabele valt, wordt de nieuwe waarde de waarde van het controlepunt. Deze waarde verschijnt vervolgens in het verificatievenster. Zie Hoofdstuk 5 voor meer informatie over het bereik.

Gegevens evalueren en wijzigen

U kunt de waarde van een uitdrukking evalueren en wijzigen zonder dat u er een controlepunt voor hoeft in te stellen of de waarde hoeft te verifiëren. U kunt dat op verschillende manieren doen:

- Markeer de uitdrukking of verplaats de cursor in het modulevenster naar de uitdrukking in de broncode en voer een van de volgende handelingen uit:

  - Klik op **Evalueren/Wijzigen** op de knoppenbalk
  - Kies **Programma | Evalueren/Wijzigen** of rechtsklik op het modulevenster en kies **Evalueren/Wijzigen** in het snelmenu

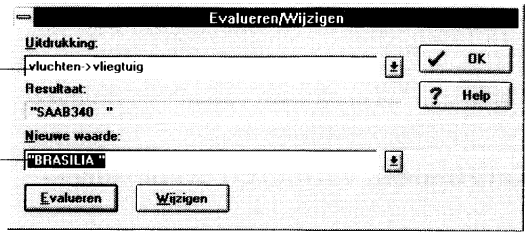
  De uitdrukking die u hebt opgegeven, verschijnt in het dialoogvenster **Evalueren/Wijzigen** (Afbeelding 8.15).

- Klik op **Evalueren/Wijzigen** op de knoppenbalk en typ de uitdrukking die u wilt evalueren in het vak **Uitdrukking** van het dialoogvenster **Evalueren/Wijzigen**.
- Kies **Programma | Evalueren/Wijzigen** of rechtsklik op het modulevenster en kies **Evalueren/Wijzigen** in het snelmenu. Typ vervolgens de uitdrukking die u wilt evalueren in het vak **Uitdrukking** van het dialoogvenster **Evalueren/Wijzigen**.

Als de uitdrukking die u wilt evalueren, zich in het vak **Uitdrukking** van het dialoogvenster **Evalueren/Wijzigen** bevindt, klikt u op de knop **Evalueren** als u de huidige waarde van de uitdrukking wilt weergeven in het vak **Resultaat**.

In het dialoogvenster **Evalueren/Wijzigen** kunt u de huidige waarde van de uitdrukking wijzigen door een nieuwe waarde te typen in het vak **Nieuwe waarde** en vervolgens op de knop **Wijzigen** te klikken. U kunt het gegevenstype van een uitdrukking niet wijzigen.

Afbeelding 8.15 Het dialoogvenster Evalueren/Wijzigen



Klik op de knop Evalueren als u de huidige waarde wilt weergeven van de uitdrukking die u hebt getypt in het vak Uitdrukking

Typ een nieuwe waarde voor de uitdrukking in het vak Nieuwe waarde. Klik op de knop Wijzigen als u de waarde wilt wijzigen.

De debugger configureren

Kies **Opties** in de menubalk van de debugger als u opties voor de omgeving van de debugger wilt instellen, als u deze instellingen wilt opslaan in een .CFG-bestand of als u de instellingen wilt herstellen die zijn opgeslagen in een bestaand .CFG-bestand. U kunt verscheidene .CFG-bestanden maken met in elk bestand verschillende configuratie-instellingen. Op die manier kunt u altijd de gewenste instellingen gebruiken voor de huidige debug-sessie.

Standaardinstellingen voor de configuratie opgeven

Als u de configuratie-opties wilt opgeven die u in een .CFG-bestand wilt opslaan, kiest u **Opties | Standaard**. In het dialoogvenster **Beginwaarden** kunt u de volgende standaardinstellingen opgeven:

- **Bureaublad**. De rangschikking van de vier debugger-vensters en de positie van de knoppenbalk.
- **Applicatie**. De applicatie die u standaard wilt debuggen.
- **Afbreekpunten**. De afbreekpunten die op dat moment zijn ingesteld in het afbreekpuntenvenster en die hierin worden weergegeven.
- **Controlepunten**. De controlepunten die op dat moment zijn ingesteld in het controlevenster en die hierin worden weergegeven.

U kunt de knoppenbalk van de debugger uitschakelen of de positie ervan wijzigen met de volgende keuzerondjes in het dialoogvenster **Beginwaarden**:

- **Normaal**
- **Popup**
- **Verkleind**
- **Verticaal**

De animatiesnelheid instellen

Als u een standaard animatiesnelheid wilt instellen, kiest u **Opties | Animatiesnelheid**. In het dialoogvenster **Animatiesnelheid instellen** verplaatst u het schuifblokje op de schuifbalk naar rechts als u de animatiesnelheid wilt verhogen en naar links als u de animatiesnelheid

wilt verlagen. Als u een programma animeert en u wijzigt de animatiesnelheid, wordt de snelheid van de huidige animatie dienovereenkomstig aangepast.

Font instellen

Als u standaard weergavefonts voor tekst in de debugger wilt instellen, kiest u **Opties | Font weergeven**. Geef de gewenste voorkeuren op in het dialoogvenster **Font**.

Pad voor bronbestanden instellen

Als u een standaardpad naar het station en de directory wilt opgeven van waaruit bestanden in de debugger moeten worden geladen, kiest u **Opties | Pad bronbestand**. Typ het gewenste pad in het dialoogvenster **Pad bronbestand**.

Configuratie-instellingen opslaan in .CFG-bestanden

Als u de instellingen wilt opslaan die u opgeeft met **Standaard**, **Animatiesnelheid**, **Fonts weergeven** en **Pad bronbestand** in het menu **Opties**, kiest u **Opties | Opties opslaan**. Geef in het dialoogvenster **Configuratie opslaan** de naam op van het configuratiebestand. Standaard wordt de bestandsnaam DEFAULT.CFG voorgesteld.

Als u de standaard bestandsnaam DEFAULT.CFG accepteert, wordt dit bestand gebruikt als u de debugger start. Als u een ander .CFG-bestand wilt gebruiken, moet u dit bestand laden nadat u de debugger hebt gestart. Zie voor meer informatie de volgende sectie, "Configuratie-instellingen herstellen". Als het bestand DEFAULT.CFG niet wordt gevonden, worden de oorspronkelijke instellingen gebruikt.

Configuratie-instellingen herstellen

Totdat u een .CFG-bestand maakt of herstelt, gebruikt de debugger het systeemfont van Windows, de huidige directory en de hoogste animatiesnelheid. Bovendien worden alle vensters weergegeven en bevindt de knoppenbalk zich onder de menubalk. Er worden geen standaard-afbreekpunten, -controlepunten of -applicaties ingesteld.

Als het bestand DEFAULT.CFG wordt aangetroffen, wordt dit gebruikt. U kunt de instellingen herstellen die u in een ander .CFG-bestand dan DEFAULT.CFG hebt opgeslagen door **Opties | Opties herstellen** te kiezen. Geef in het dialoogvenster **Configuratie herstellen** de naam op van het .CFG-bestand met de instellingen die u wilt herstellen.

De debugger verbinden met dBASE-instanties

U kunt meerdere instanties van dBASE en de debugger starten. Als u de debugger vanuit dBASE start, wordt de debugger verbonden met de dBASE-instantie waarin deze werd gestart. Als u echter meerdere dBASE-instanties hebt gestart, kunt u verschillende applicaties uitvoeren in de verschillende instanties en kunt u opgeven met welke applicatie in welke instantie u de debugger wilt verbinden. Hiertoe gaat u als volgt te werk:

1 Kies **Bestand | Verbinden.**

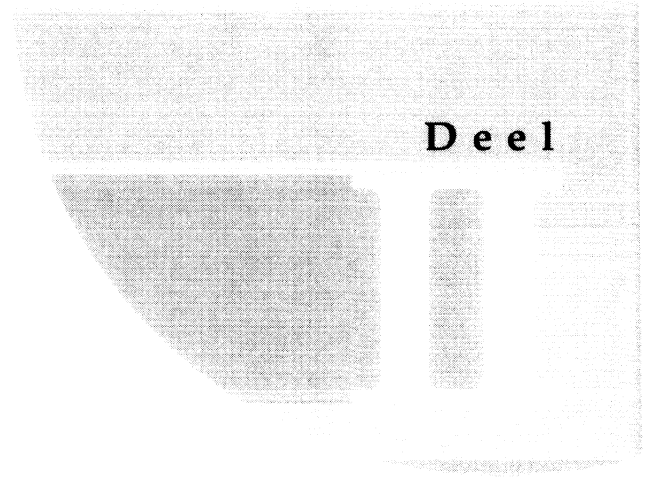
2 Geef in het dialoogvenster **Verbinden aan actieve applicatie** de applicatie in de huidige dBASE-instantie op waarmee u de debugger wilt verbinden.

De huidige directory wijzigen

U kunt de directory voor de huidige debug-sessie wijzigen door **Bestand | Directory wijzigen** te kiezen. Geef in het dialoogvenster **Directory wijzigen** het pad naar de nieuwe directory op in het vak **Nieuwe directory.**

Programmabestanden weergeven

Als u een programmabestand (of een willekeurig tekstbestand) wilt weergeven terwijl de debugger actief is, kiest u **Bestand | Tekstbestand openen.** Het bestand wordt geopend in een venster dat op de voorgrond wordt weergegeven. U kunt in het bestand schuiven, maar u kunt het bestand niet bewerken.



Objecten en klassen

Dit deel bevat een inleiding en bespreking van de objectgeöriënteerde uitbreidingen van de dBASE-taal.

Is dit uw eerste kennismaking met objectgeöriënteerd programmeren, dan begint u met het volgende hoofdstuk:

1 Hoofdstuk 9, over achterliggende concepten en terminologie

Vervolgens kunt u een eerste blik werpen op de volgende hoofdstukken:

1 Hoofdstuk 10 en 11, over commando's en technieken

2 Hoofdstuk 12, over het ontwerpen van objectgeoriënteerde applicaties

Begint u eenmaal met programmeren met behulp van objecten en klassen, dan raadpleegt u Hoofdstuk 10 en 11 voor meer informatie over bepaalde taken:

Dit deel bevat de volgende hoofdstukken:

- Hoofdstuk 9, "Inleiding tot object-georiënteerd programmeren"
- Hoofdstuk 10, "Werken met objecten"
- Hoofdstuk 11, "Werken met klassen"
- Hoofdstuk 12, "Object-georiënteerd ontwerpen"

Hoofdstuk

9

Inleiding tot object-georiënteerd programmeren

In dit hoofdstuk wordt een inleiding gegeven tot het werken met objecten en klassen en wordt beschreven hoe u hiervoor de dBASE-taal gebruikt. U leert wat objecten en klassen zijn, hoe u objecten maakt en hoe u objecten manipuleert door de kenmerken ervan te wijzigen.

In object-georiënteerd programmeren worden geavanceerde technieken gebruikt voor de afhandeling van geheugenvariabelen. Voordat u leert werken met objecten, dient u bekend te zijn met de grondbeginselen van het gebruik van geheugenvariabelen. Zie Hoofdstuk 5 voor meer informatie.

Een object-georiënteerde kennismaking

Met behulp van de object-georiënteerde functies van dBASE voor Windows kunt u met eenvoudige toewijzingsopdrachten ("dit" is gelijk aan "dat") hetzelfde bereiken als waar u in vroegere versies van dBASE verscheidene regels code voor nodig had. Dit wordt *programmeren met kenmerken genoemd.* Dat wil zeggen, programmeren door de waarden te wijzigen van de kenmerken van een object. Als u wilt weten hoe gemakkelijk deze manier van programmeren is, volgt u de voorbeelden in deze sectie waarin een formulierobject wordt gemaakt en waarin de bijbehorende kenmerken worden gemanipuleerd.

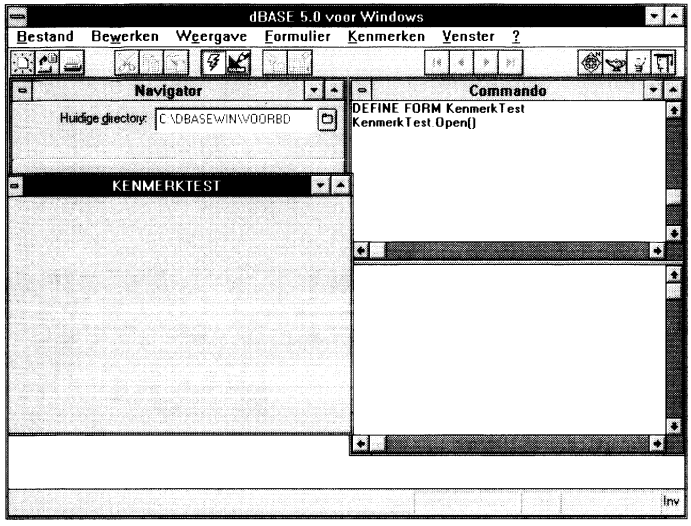
Eerst maakt en opent u een formulierobject door de volgende twee regels in het commandovenster te typen:

```
DEFINE FORM KenmerkTest  && Formulier maken
KenmerkTest.Open()       && Nieuw formulier openen
```

U hebt zojuist een "instantie" of exemplaar van de klasse Form gemaakt, met de naam *KenmerkTest*. Programmeren met kenmerken wordt ook wel eens *programmeren met*

instanties genoemd, omdat u een instantie van een klasse dynamisch manipuleert. *KenmerkTest* ziet er als volgt uit:

Afbeelding 9.1 Het formulier KenmerkTest

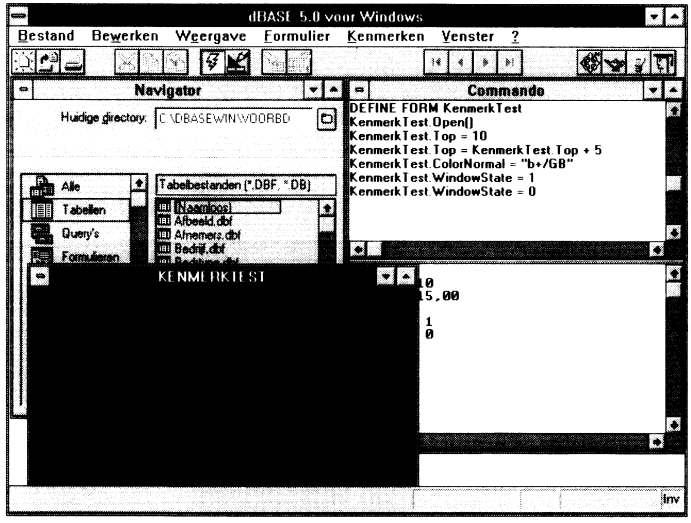


De reden waarom programmeren met kenmerken een bijzonder krachtige programmeermethode is in dBASE, is dat als u de waarde van een kenmerk wijzigt, dit onmiddellijk wordt weergegeven door het object. Als u wilt weten hoe dit in zijn werk gaat, typt u de volgende regels met code. U ziet dan hoe het formulier *KenmerkTest* wordt gewijzigd (u kunt de commentaarregels eventueel weglaten):

```
KenmerkTest.Top = 10                    && Formulier verplaatsen naar rij 10
KenmerkTest.Top = KenmerkTest.Top + 5   && Formulier 5 rijen extra verplaatsen
KenmerkTest.ColorNormal = "b+ GB"       && Kleur wijzigen
KenmerkTest.WindowState = 1             && Formulier verkleinen tot pictogram
KenmerkTest.WindowState = 0             && Positie van formulier herstellen
```

KenmerkTest ziet er nu als volgt uit:

Afbeelding 9.2 Gewijzigd formulier KenmerkTest



Objecten kunnen worden uitgebreid. Dat wil zeggen, met een eenvoudige toewijzingsopdracht kunt u nieuwe elementen aan een object toevoegen. Typ de volgende regels in het commandovenster als u twee nieuwe kenmerken wilt toevoegen aan het formulier:

```
KenmerkTest.MijnKenmerkN = 123                  && Nieuw kenmerk MijnKenmerkN toevoegen
KenmerkTest.MijnKenmerkC = "Hallo!"             && Nieuw kenmerk MijnKenmerkC toevoegen
```

Deze toevoeging van kenmerken heeft nog geen invloed op het formulier. Als u wilt dat een kenmerk een bewerking op het formulier uitvoert, kunt u een methode toevoegen waarmee de waarde van het kenmerk wordt gelezen en een handeling met het kenmerk wordt uitgevoerd. In het volgende voorbeeld wordt een methode toegevoegd aan *KenmerkTest* door een codeblok toe te wijzen aan *MijnMethode*. Typ de volgende regel in het commandovenster:

```
KenmerkTest.MijnMethode = {;KenmerkTest.Text = KenmerkTest.text+KenmerkTest.MijnKenmerkC}
```

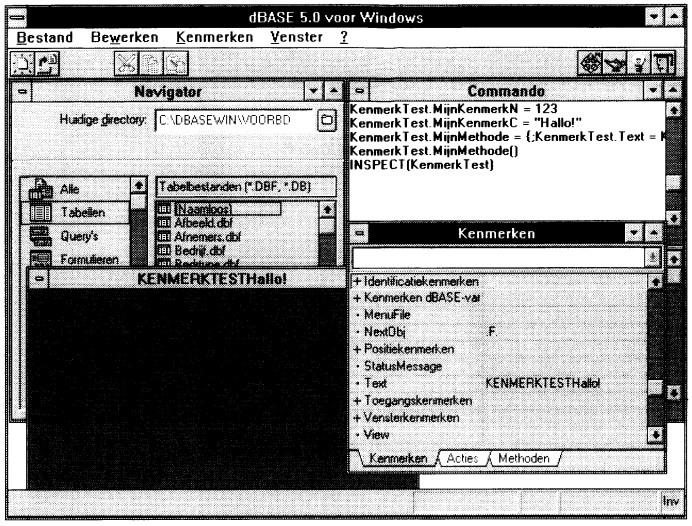
Als *MijnMethode* wordt uitgevoerd, wordt de reeks die is opgeslagen in *MijnKenmerkC*, toegevoegd aan de titel van het formulier. Als u *MijnMethode* wilt uitvoeren, gebruikt u de aanroepingsoperator als volgt:

```
KenmerkTest.MijnMethode()                       && Titel wijzigen
```

Kenmerken instellen door commando's te typen is een handige methode, maar soms is het gemakkelijker om alle kenmerken van een object interactief weer te geven en te bewerken. U doet dit in het kenmerkenvenster. Open het venster als volgt:

```
INSPECT(KenmerkTest)                            && Kenmerkenvenster weergeven
```

Afbeelding 9.3 Het kenmerkenvenster van KenmerkTest



dBASE bevat een systeemgeheugenvariabele, _app genaamd, waarmee u het applicatie-object kunt benaderen. In het applicatie-object kunt u verschillende kenmerken van dBASE instellen. Typ de volgende regel in het commandovenster:

```
? _app.FrameWin.Text                    && Titel van applicatievenster weergeven
```

Typ nu de volgende regel:

```
_app.FrameWin.Text = "Mijn eigen"+_app.FrameWin.Text
```

U hebt zojuist dBASE voorzien van uw eigen titel!

Nu u een idee hebt gekregen hoe u met objecten werkt, leest u de volgende secties over objecten om te zien hoe objecten in dBASE worden geïmplementeerd.

Over objecten

In Formulierontwerp maakt en manipuleert u objecten voor gebruikersinterfaces met behulp van interactieve hulpmiddelen zoals kenmerkenvensters. Als programmeur gebruikt u objecten als verzamelingen geheugenvariabelen. Het object zelf is een container voor geheugenvariabelen, waarvan sommige gegevenswaarden bevatten en andere verwijzende subroutines.

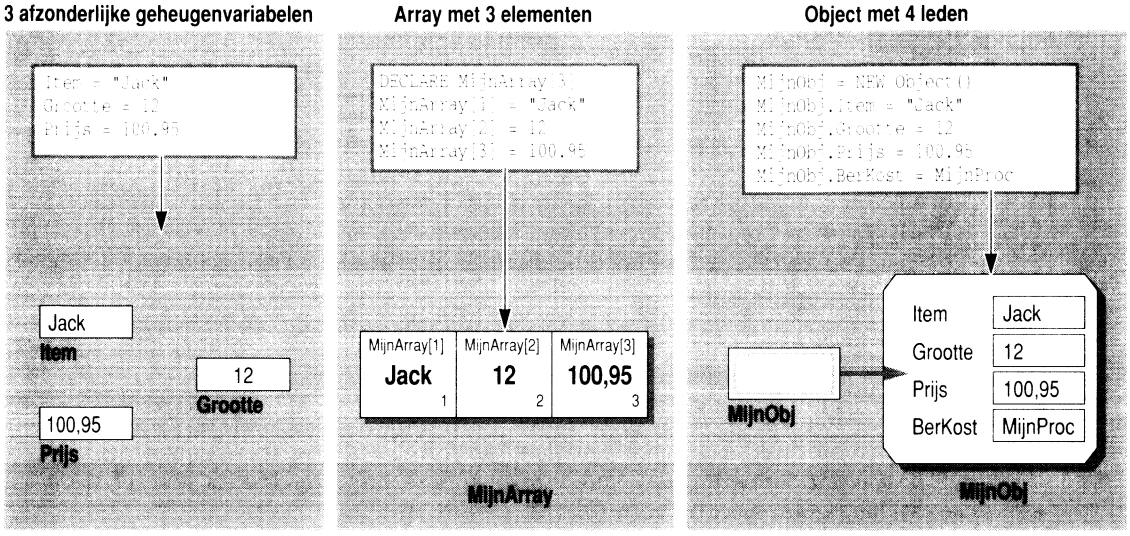
Als u al eerder met array's hebt gewerkt, kunt u een object ook beschouwen als een array, waarbij elk objectkenmerk correspondeert met een element in een array. (In feite *is* een array in dBASE een soort object. Zie "Werken met array's als objecten" in Hoofdstuk 10 voor meer informatie.)

- Een array is een verzameling variabelen die sequentieel zijn ingedeeld in rijen en kolommen. U verwijst naar array-elementen met getallen die de positie van de elementen in de array aangeven. `MijnArray[1]` duidt het eerste element in de array aan en `MijnArray[3,2]` het tweede element in de derde rij.

- Een object is een verzameling variabelen zonder impliciete organisatie. De variabelen in een object zijn *leden* van het object. U verwijst naar de leden van een object met de naam van het desbetreffende object. `MijnObj.Item` verwijst naar het kenmerk Item en `MijnObj.Prijs` verwijst naar het kenmerk Prijs.

In Afbeelding 9.4 wordt de relatie tussen afzonderlijke geheugenvariabelen, array's en objecten toegelicht en wordt de code weergegeven waarmee u deze maakt.

Afbeelding 9.4 Een object is een verzameling geheugenvariabelen



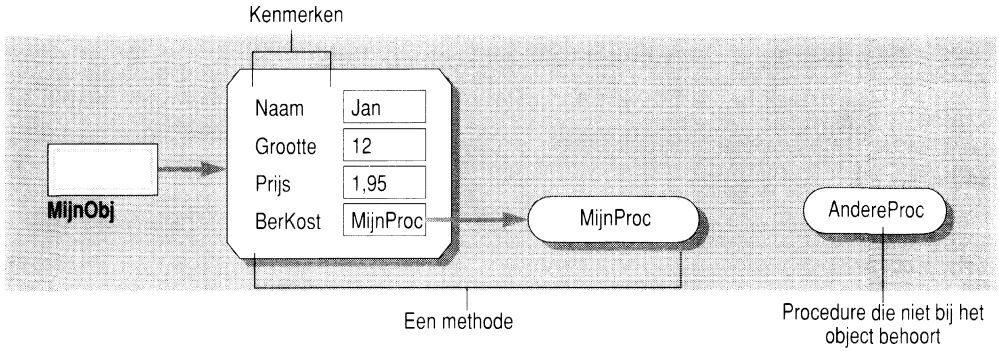
Alle geheugenvariabelen in een object zijn kenmerken van het object. Sommige kenmerken veranderen in *methoden* wanneer u er code aan verbindt.

Een methode is een subroutine, zoals een procedure, functie of codeblok, die met een object is verbonden door middel van een functie-aanwijzer. Met methoden voert u handelingen uit op een object. Als u bijvoorbeeld een formulier wilt sluiten, kunt u de methode Close van het formulier aanroepen. (In Hoofdstuk 3 worden functie-aanwijzers en codeblokken beschreven.)

Sommige methoden in ingebouwde objecten, zoals een formulier of invoervak, worden automatisch uitgevoerd in reactie op een gebeurtenis, zoals een muisklik. Deze methoden worden *acties* genoemd omdat de uitvoering van deze methoden wordt ingeleid door een "actie". Als de gebruiker bijvoorbeeld op een knopobject klikt, wordt de actie OnClick van de knop uitgevoerd. Andere methoden worden alleen uitgevoerd wanneer ze expliciet worden aangeroepen.

In Afbeelding 9.5 ziet u een object dat is opgebouwd uit kenmerken, waarvan sommige methoden zijn.

Afbeelding 9.5 De leden van een object



De variabelen in een object (de leden van het object) kunnen van hetzelfde gegevenstype zijn als gewone variabelen. In de voorgaande voorbeelden heeft het kenmerk Naam het gegevenstype Teken. Maat en Prijs zijn numerieke kenmerken. Het gegevenstype van BerKost is Functie-aanwijzer. BerKost bevat zelf geen waarde, maar verwijst naar een subroutine.

Objecten en hoofdobjecten

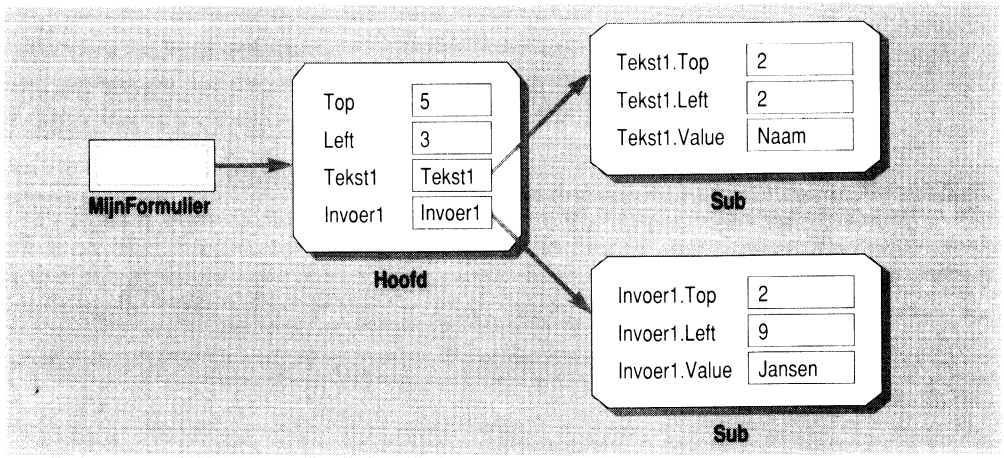
Objecten kunnen andere objecten bevatten. Formulieren zijn de bekendste voorbeelden van objecten die andere objecten bevatten. Het formulierobject is het containerobject, ofwel *hoofdobject*. Invoervakken, knoppen en andere interface-objecten zijn *subobjecten* waarvan het hoofdobject het formulier is.

Met de volgende code maakt u een formulierobject (MijnFormulier), een tekstobject (Tekst1) en een invoervakobject (Invoer1):

```
DEFINE FORM MijnFormulier PROPERTY Top 5, Left 3
DEFINE TEXT Tekst1 OF MijnFormulier PROPERTY Top 2, Left 2, Text "Naam"
DEFINE ENTRYFIELD Invoer1 OF MijnFormulier PROPERTY Top 2, Left 9, Value "Jansen"
OPEN FORM MijnFormulier
```

In Afbeelding 9.6 ziet u het resultaat van deze code:

Afbeelding 9.6 Hoofdobjecten en subobjecten



Top en Left zijn numerieke kenmerken van MijnFormulier. Tekst1 en Invoer1 zijn van het type Object en verwijzen dus naar een ander object.

```
? MijnFormulier                    && Variabele van type Object
? MijnFormulier.Tekst1             && Eveneens variabele van type Object
? MijnFormulier.Invoer1            && Eveneens variabele van type Object
```

De hoofd-sub-relatie tussen objecten heeft belangrijke gevolgen voor de manier waarop u met objecten werkt. Als een hoofdobject wordt vrijgegeven, worden alle bijbehorende subobjecten eveneens vrijgegeven. In de volgende code ziet u dat zowel Tekst1 als Invoer1 worden vrijgegeven wanneer u MijnFormulier vrijgeeft.

```
RELEASE OBJECT MijnFormulier       && Het formulier verdwijnt
? MijnFormulier                    && Variabele bestaat, maar verwijst niet naar object
? MijnFormulier.Tekst1             && Variabele niet-gedefinieerd
? MijnFormulier.Invoer1            && Variabele niet-gedefinieerd
```

Elk object dat u maakt, is gebaseerd op een klasse. MijnObj in Afbeelding 9.4 en Afbeelding 9.5 is gebaseerd op de klasse Object. MijnFormulier, Tekst1 en Invoer1 in Afbeelding 9.6 zijn respectievelijk gebaseerd op de klasse Form, Text en EntryField. Maar wat is nu precies een klasse?

Over klassen

Een *klasse* is een specificatie, of sjabloon, voor het maken van meerdere kopieën van een bepaald object. Zonder klassen zou u de kenmerken van elk nieuw object opnieuw moeten definiëren wanneer u het object maakt. U gebruikt klassen als "objectfabrieken" als u identieke kopieën van objecten wilt maken. Het is bijvoorbeeld mogelijk dat u een eigen knop "Zoeken" gebruikt in meerdere formulieren. In plaats van deze knop telkens opnieuw te maken, kunt u een klasse Zoeken definiëren waarmee u de knop definieert. Vervolgens kunt u een willekeurig aantal exemplaren van deze klasse maken.

dBASE bevat vele ingebouwde klassen, zoals Form en Entryfield. Als u een formulier maakt in Formulierontwerp, maakt u een nieuw formulierobject op basis van de ingebouwde specificaties uit de klasse Form.

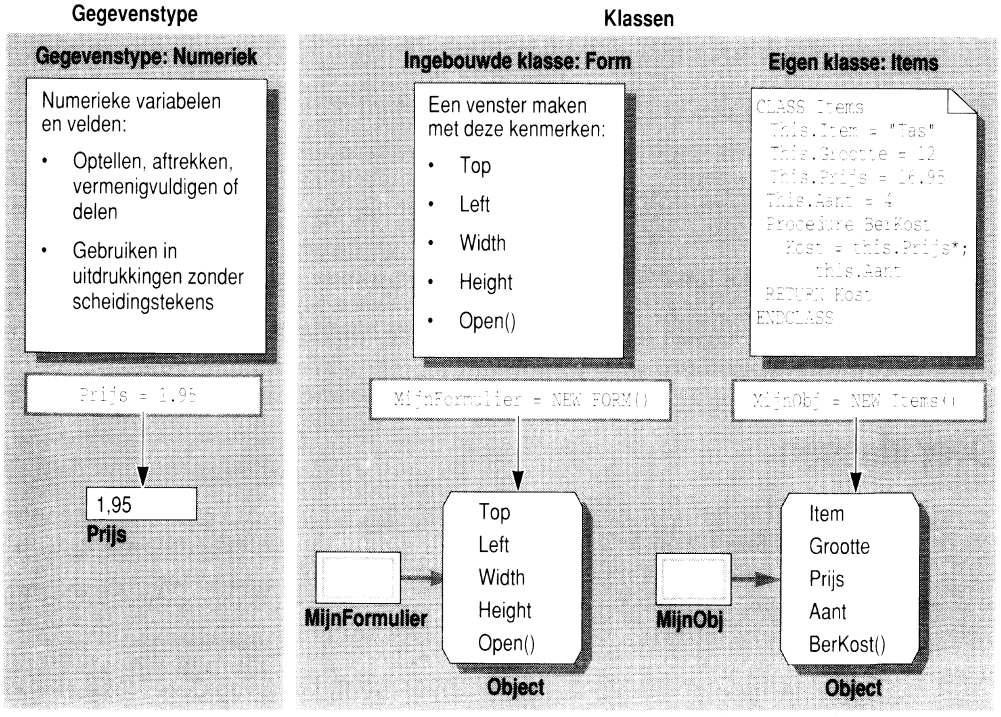
De relatie tussen klasse en object lijkt op de relatie tussen een gegevenstype en een geheugenvariabele. Het gegevenstype Numeriek bevat bijvoorbeeld regels voor gegevens van het type Numeriek en de manier waarop u deze gegevens manipuleert. Als u een numerieke variabele maakt, maakt u een exemplaar van het gegevenstype Numeriek. U kunt zoveel numerieke variabelen maken als u wilt en voor elke variabele een andere waarde gebruiken. De regels die bepalen hoe u numerieke variabelen kunt manipuleren, zijn echter voor alle numerieke variabelen hetzelfde.

Op dezelfde wijze bevat de klasse Form de karakteristieken van alle formulieren. Als u een formulier maakt, maakt u een exemplaar van de klasse Form. U kunt zoveel formulieren maken als u wilt, en in elk formulier kunt u verschillende gegevens weergeven. Alle formulieren hebben echter de karakteristieken die zijn vastgelegd in de klasse Form.

Als u een eigen klasse zoals de klasse "Zoeken" van hierboven definieert, geeft u *zelf* de kenmerken en methoden van objecten van deze klasse op met dBASE-code.

In Afbeelding 9.7 wordt de relatie tussen gegevenstypen en variabelen en tussen klassen en objecten weergegeven.

Afbeelding 9.7 Een klasse is voor een object wat een gegevenstype is voor een geheugenvariabele



Samenvatting van object-georiënteerde onderwerpen

In dit hoofdstuk hebt u geleerd hoe u objecten manipuleert met behulp van programmeren met kenmerken en hebt u kennis gemaakt met de grondbeginselen van objecten en klassen. In de twee volgende hoofdstukken wordt uitgebreid beschreven hoe u met objecten en klassen werkt. Hierna volgt een beknopte samenvatting van de wijze waarop u objecten en klassen in dBASE kunt gebruiken:

Objecten

- Objecten maken op basis van ingebouwde klassen en klassen die u zelf definieert
- Een query uitvoeren op de waarde van een kenmerk en de waarde van de meeste kenmerken wijzigen
- Nieuwe kenmerken aan een object toevoegen met een eenvoudige toewijzingsopdracht
- Onmiddellijk de resultaten weergeven wanneer u de waarde wijzigt van een kenmerk van een ingebouwd object

Klassen

- Eigen klassen definiëren
- Klassen definiëren die zijn gebaseerd op andere klassen en die daarvan de kenmerken en methoden overnemen
- Een hiërarchie van klassen samenstellen

Hoofdstuk

10

Werken met objecten

In Hoofdstuk 9 hebt u geleerd wat objecten en klassen zijn en waarom dit krachtige programmeermiddelen zijn. In dit hoofdstuk worden de grondbeginselen behandeld van het maken en gebruiken van objecten, worden hoofd- en subobjecten beschreven en wordt aangegeven hoe u hiërarchieën van objecten gebruikt. Het hoofdstuk wordt afgesloten met taken voor gevorderden, zoals het maken van eigen objecten en het werken met array's als objecten.

Objecten maken

Als u een geheugenvariabele wilt maken, definieert u een naam. In dit voorbeeld wordt een variabele van het type LOCAL gemaakt met de naam nPrijs:

```
LOCAL nPrijs
nPrijs = 1.95
```

Als u een array wilt maken, definieert u de naam van de array en het aantal rijen en kolommen in de array. In dit voorbeeld wordt een array met 5 rijen en 2 kolommen gemaakt. De naam van de array is MijnArray:

```
DECLARE MijnArray[5,2]
MijnArray[1,1] = "Eerste element"
```

Als u een object wilt maken, definieert u de naam van het object en geeft u de klasse op waarop het object moet worden gebaseerd. Een klasse is een soort sjabloon die de kenmerken en methoden bevat waaruit een object is opgebouwd. Een object is een *exemplaar* van een klasse. Zo is een formulier een exemplaar van de klasse Form. In de klasse Form is vastgelegd dat formulierobjecten onder meer de kenmerken Height en Width hebben voor het bepalen van het formaat.

U kunt objecten op twee verschillende manieren met de dBASE-taal maken:

- Gebruik DEFINE als u met één commando een object wilt maken en de beginwaarden van de objectkenmerken wilt instellen. Programmeurs die bekend zijn

met de dBASE IV-commando's DEFINE, vinden deze syntaxis mogelijk het gemakkelijkst.

- Gebruik de operator NEW in een toewijzingsopdracht voor een geheugenvariabele. Met de operator NEW maakt u een object, maar geeft u geen waarden op voor de kenmerken van het object. Als u waarden aan kenmerken wilt toewijzen, gebruikt u aanvullende toewijzingsopdrachten.

  Doorgaans gebruikt u de operator NEW om een object te maken waarvan u de kenmerken niet hoeft te wijzigen. Een .WFM-bestand dat is gegenereerd in Formulierontwerp bevat bijvoorbeeld een eigen klasse voor een formulier met alle instellingen die u hebt opgegeven toen u het formulier ontwierp. Het .WFM-bestand bevat een regel waarmee op basis van de klasse een formulierobject wordt gemaakt met behulp van de operator NEW.

  Stel dat EIGNFORM.WFM een eigen formulier is dat is gegenereerd in Formulierontwerp. Op de volgende manier maakt u een formulierobject op basis van het .WFM-bestand:

```
DO EIGNFORM.WFM                 && Opdracht met operator NEW in .WFM-bestand uitvoeren
```

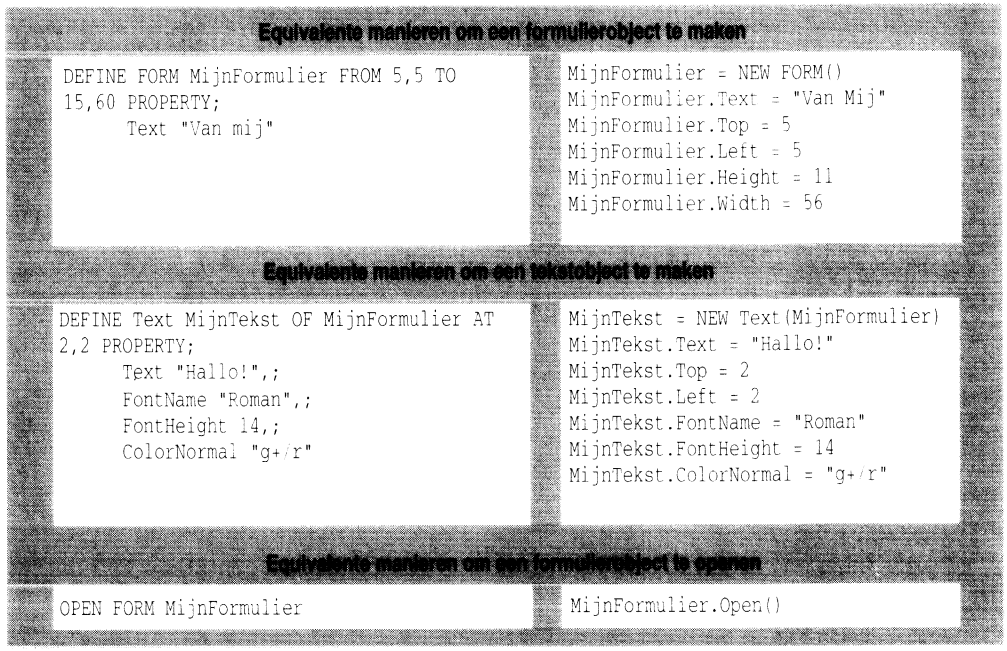
  of

```
SET PROCEDURE TO EIGNFORM.WFM   && Klassedefinitie in geheugen laden
oForm = NEW EIGNFORM()          && Formulierobject maken op basis van klassedefinitie
```

  Programmeurs die bekend zijn met andere object-georiënteerde talen, zoals Borland C++ of Pascal, zullen in de operator NEW een middel zien om objecten te maken. Mogelijk vinden zij NEW eenvoudiger in het gebruik dan DEFINE.

Hierna volgen twee codevoorbeelden, naast elkaar weergegeven, waarin u equivalente manieren ziet om een formulierobject te maken:

Afbeelding 10.1 Equivalente manieren om objecten te maken: de opdracht DEFINE en de operator NEW



Deze syntaxis is uitwisselbaar. U kunt zonder problemen een van deze vormen van syntaxis gebruiken. In een opdracht DEFINE kunt u bijvoorbeeld het object maken en maar een aantal kenmerken instellen.

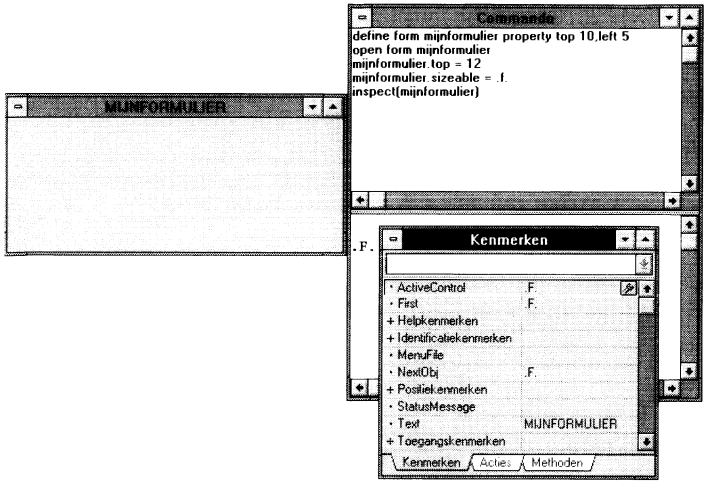
```
DEFINE FORM MijnFormulier PROPERTY Top 10, Left 5
OPEN FORM MijnFormulier
```

Vervolgens kunt u de stipnotatie gebruiken (deze notatie wordt beschreven in "De stip-operator", verderop in dit hoofdstuk) als u enkele kenmerken wilt wijzigen.

```
MijnFormulier.Top = 12
MijnFormulier.Sizeable = .F.
```

Daarna kunt met behulp van INSPECT() het kenmerkenvenster openen en kenmerken interactief wijzigen.

Afbeelding 10.2 Het kenmerkenvenster



Als u nog een object aan het formulier wilt toevoegen, gebruikt u DEFINE of de operator NEW. In dit voorbeeld wordt de operator NEW gebruikt:

```
MijnKnop = NEW PUSHBUTTON(MijnFormulier)
```

Verwijzen naar leden van een object

De syntaxis voor verwijzingen naar leden van een object is een uitbreiding van de manier waarop u verwijst naar geheugenvariabelen en array-elementen. U verwijst naar geheugenvariabelen door de naam van de variabele te gebruiken in een expressie. U verwijst naar array-elementen door de naam van de array te gebruiken, gevolgd door het indexteken van het element. In de volgende code wordt hiervan een voorbeeld gegeven:

```
LOCAL cNaam                                  && Geheugenvariabele definiëren
cNaam = "Jan"                                && Waarde toewijzen
REPLACE VoorNaam WITH cNaam                  && Verwijzen naar variabele in uitdrukking

DECLARE MijnArray[10,10]                     && Array definiëren
MijnArray[1,1] = "Stoel"                     && Waarde toewijzen aan element
? "Het bestelde item is een ",MijnArray[1,1] && Verwijzen naar element in uitdrukking
```

Als u wilt verwijzen naar leden van een object, gebruikt u de naam van het kenmerk of de methode, voorafgegaan door een *objectverwijzing*. Met een objectverwijzing geeft u het object op waarvan een kenmerk of methode een lid is. Mogelijke objectverwijzingen zijn:

- De naam van een variabele die naar het object verwijst. Dit wordt een *objectverwijzingsvariabele genoemd*. Als u een object maakt (met DEFINE of de operator NEW), wordt een objectverwijzingsvariabele met dezelfde naam gemaakt die naar

het object verwijst. In de sectie "Werken met objectverwijzingsvariabelen", op bladzijde 137, worden deze variabelen uitgebreid beschreven.

```
DEFINE FORM MijnFormulier            && Formulierobject MijnFormulier maken
MijnFormulier.top = 3                && Objectnaam gebruiken als verwijzing
MijnFormulier.ColorNormal = "w+/gb"
MijnFormulier.Open()
```

- *This.* Een ingebouwde verwijzing naar het object waarvan het kenmerk een lid is. In dBASE voor Windows wordt automatisch een *this*-verwijzing gemaakt als een klasse wordt gedefinieerd of een actie-afhandelingsroutine wordt uitgevoerd. Gebruik *this* in klassedefinities en in actie-afhandelingsroutines als u algemeen naar objecten wilt verwijzen zonder een feitelijke naam in de code op te nemen. In het volgende voorbeeld wordt hiervan een voorbeeld gegeven:

```
*** Programma DemoThis.prg
SET PROCEDURE TO PROGRAM(1) ADDITIVE
DEFINE FORM MijnFormulier
MijnFormulier.OnGotFocus = MijnOnGotFocus
MijnFormulier.Open()

Procedure MijnOnGotFocus
   this.text = "Hé! Ik heb de focus!"
RETURN
```

- *Form.* Een ingebouwde verwijzing naar het formulier dat een object bevat. In dBASE wordt automatisch een *form-verwijzing* gemaakt als u een formulierklasse definieert of een formulier opent. Evenals *this* gebruikt u *form* in code voor actie-afhandelingsroutines en klassedefinities als u algemeen wilt verwijzen naar het formulier dat het object bevat. In het volgende programma wordt hiervan een voorbeeld gegeven:

```
*** Programma DemoForm.prg
SET PROCEDURE TO PROGRAM(1) ADDITIVE
DEFINE FORM MijnFormulier
DEFINE PUSHBUTTON MijnKnop OF MijnFormulier
MijnFormulier.MijnKnop.OnClick = MijnOnClick

Procedure MijnOnClick
   Form.text = "Hé! U hebt op mijn knop gedrukt!"
RETURN
```

This en *form* komen uitgebreid aan de orde in Hoofdstuk 14.

In de voorgaande voorbeelden is de naam van het object gescheiden van de naam van het kenmerk of de methode door een *lid-toegangsoperator*. De lid-toegangsoperator verbindt een lid met een object op ongeveer dezelfde wijze als de alias-operator (->) een veld met een tabel verbindt. De lid-toegangsoperatoren zijn de stip-operator en de index-operator. Deze operatoren worden in de volgende sectie beschreven.

De stip-operator

De *stip-operator* (.) verwijst naar leden van een object met hun naam. De syntaxis is als volgt:

```
<objectverwijzing>.<lidnaam>
```

De stip-operator is de meest gebruikelijke manier om te verwijzen naar objectkenmerken of methoden. Commandosyntaxis met de stip-operator wordt vaak *stipnotatie genoemd.*

```
DEFINE FORM MijnFormulier            && Formulierobject maken
MijnFormulier.Left = 200             && Kenmerk Left van MijnFormulier instellen
MijnFormulier.Text = "Hallo"         && Kenmerk Text van MijnFormulier instellen
```

Het resultaat van de stip-operator kan weer een objectverwijzing zijn, die verwijst naar een subobject. U kunt naar kenmerken van subobjecten verwijzen door een pad van objectverwijzingen samen te stellen die naar het kenmerk leiden. Dit is vergelijkbaar met het opgeven van een pad naar een bestand op schijf, waarbij u elke subdirectory opgeeft die naar het bestand leidt.

```
DEFINE PUSHBUTTON MijnKnop OF MijnFormulier && Knop in formulier maken
MijnFormulier.MijnKnop.Top = 2              && Kenmerk Top van MijnKnop in MijnFormulier
                                            && instellen
```

De index-operator

De *index-operator* ([ ]) verwijst naar leden van een object met getallen. De syntaxis is als volgt:

```
<objectverwijzing>[<Nuitdr>]
```

Gebruik de index-operator als u wilt verwijzen naar een lid waarvan de lokatie wordt weergegeven door een waarde in plaats van een naam. In de meeste gevallen gebruikt u de index-operator om te verwijzen naar elementen van array-objecten. De naam "index-operator" is terug te voeren op array's: de vierkante haken sluiten de indexlokatie in van een element in de array. In dBASE worden array's geïmplementeerd als een speciaal soort object.

Zie Hoofdstuk 5 voor voorbeelden van het gebruik van de index-operator met array's. Zie "Minimale array's maken", verderop in dit hoofdstuk, voor voorbeelden van het gebruik van de index-operator met minimale array's die zijn gemaakt op basis van de klasse Object.

In het volgende voorbeeld ziet u hoe u de index-operator gebruikt met een object van de klasse Object.

```
MijnObj = NEW Object()
MijnObj[1] = "De lokatie van dit kenmerk is: 1"
MijnObj[5] = "De lokatie van dit kenmerk is: 5"
? MijnObj[1]
? MijnObj[5]
```

Eigen kenmerken en methoden toevoegen

Naast de ingebouwde leden van een object kunt u uw eigen kenmerken en methoden maken. U maakt eigen kenmerken en methoden als volgt:

- Gebruik het commando DEFINE met de optie CUSTOM. Met de optie CUSTOM markeert u het begin van een definitielijst van een eigen kenmerk. Op dezelfde wijze markeert u met de optie PROPERTY het begin van een lijst met ingebouwde kenmerken. Hanteer voor eigen kenmerken dezelfde syntaxis als voor ingebouwde kenmerken.
- Gebruik de operator NEW en typ de naam van het nieuwe kenmerk of de nieuwe methode achter de stip-operator.

In Afbeelding 10.3 ziet u equivalente manieren om eigen kenmerken toe te voegen: eenmaal met DEFINE en eenmaal met NEW.

Afbeelding 10.3 Eigen elementen toevoegen aan een object

optie CUSTOM van DEFINE

```
DEFINE FORM KlantForm;
   PROPERTY;
      Top 5,;
      Left 10,;
   CUSTOM;
      Begroeting "Hallo",;
      Geluksgetal 19
```

operator NEW

```
KlantForm = NEW FORM()
KlantForm.Top = 5
KlantForm.Left = 10
KlantForm.Begroeting = "Hallo"
KlantForm.Geluksgetal = 19
```

Zoals u uit Afbeelding 10.3 kunt afleiden, moet u nauwkeurig te werk gaan als u kenmerken instelt met de stip-operator. Als u de naam van een objectlid verkeerd spelt in een toewijzingsopdracht, geldt evenals bij de naam van een traditionele geheugenvariabele dat dBASE ervan uitgaat dat u een nieuw eigen element met de verkeerd gespelde naam wilt toevoegen. Er wordt dan ook geen foutmelding gegeven.

```
MijnFormulier = NEW FORM()          && Formulier maken
MijnFormulier.Text = "Hallo"        && Kenmerk Left wijzigen in 10
MijnFormulier.Tekst = "Hallo"       && Oei! Nieuw kenmerk Tekst toegevoegd
```

Werken met objectverwijzingsvariabelen

Een van de belangrijkste concepten die u moet begrijpen wanneer u met objecten werkt, is het verschil tussen traditionele variabelen en objectverwijzingsvariabelen. Als u dit verschil begrijpt, is de laatste hindernis voor het werken met objecten genomen.

Traditionele geheugenvariabelen *bevatten* waarden. Dat wil zeggen, een variabele van het type Numeriek bevat een numerieke waarde en een variabele van het type Teken bevat een tekenreeks. Als u een geheugenvariabele (*Naam*) toewijst aan een andere geheugenvariabele (*Naam2*), wordt de inhoud van *Naam* gekopieerd naar *Naam2*.

Wijzigingen die u hierna aanbrengt in de inhoud van *Naam*, hebben geen invloed op *Naam2*. In het volgende voorbeeld wordt dit toegelicht:

```
Naam = "Jan"        && Naam bevat de reeks "Jan"
Naam2 = Naam        && Waarde van Naam kopiëren naar Naam2; Naam2 bevat de reeks 'Jan'
Naam = "Sandra"     && Naam bevat nu de reeks "Sandra"
? Naam              && Geeft "Sandra" als resultaat
? Naam2             && Geeft "Jan" als resultaat
```

Dit in tegenstelling tot een objectverwijzingsvariabele, die een *verwijzing* naar een object bevat, niet het object zelf. Als u de ene objectverwijzingsvariabele toewijst aan een andere, kopieert u het object niet, maar maakt u een volgende verwijzing naar hetzelfde object.

Objectverwijzingen lijken op parameters die als verwijzing worden doorgegeven in een procedure. Als u een parameter doorgeeft als verwijzing, wordt de variabele niet gekopieerd. In plaats daarvan wordt nog een verwijzing naar de variabele in de procedure gemaakt. (In Hoofdstuk 3 wordt het doorgeven van parameters beschreven.)

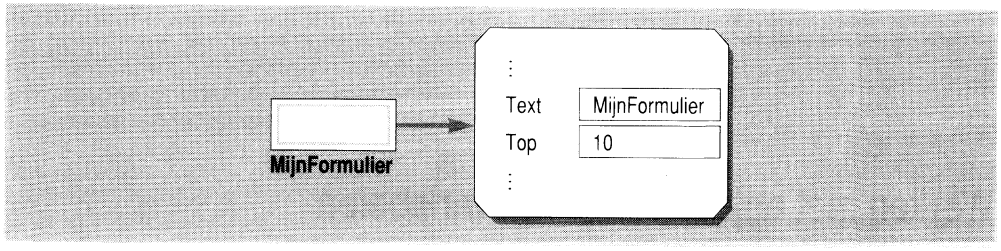
U kunt objectverwijzingsvariabelen op dezelfde manier gebruiken als andere geheugenvariabelen. U kunt objectverwijzingsvariabelen bijvoorbeeld doorgeven als parameters, teruggeven als resultaten van functies en u kunt deze variabelen opslaan in array's. In Hoofdstuk 10 worden hiervan voorbeelden gegeven.

In de volgende code wordt een formulierobject gemaakt en worden enkele van de bijbehorende kenmerken ingesteld. De objectverwijzingsvariabele MijnFormulier verwijst naar het formulier.

```
MijnFormulier = NEW Form()                  && Formulierobject waarnaar MijnFormulier verwijst
OPEN FORM MijnFormulier                     && Formulier openen
? TYPE("MijnFormulier")                     && Geeft "O" als resultaat van objectverwijzing
MijnFormulier.Text = "Mijn Formulier"       && Kenmerk Text wijzigen
MijnFormulier.Top = 10                      && Kenmerk Top wijzigen
```

In Afbeelding 10.4 wordt het resultaat van de vorige code weergegeven.

Afbeelding 10.4 MijnFormulier verwijst naar een formulierobject



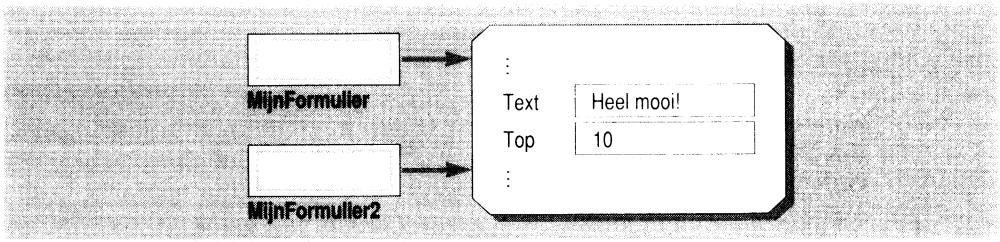
Vervolgens maakt u een andere objectverwijzingsvariabele, MijnFormulier2, en gebruikt u beide variabelen om het formulierobject te manipuleren.

```
MijnFormulier2 = MijnFormulier              && MijnFormulier en MijnFormulier2 verwijzen
                                            && naar hetzelfde formulier
? MijnFormulier2.Text                       && Geeft "Mijn Formulier" als resultaat
MijnFormulier2.Text = "Heel mooi!"          && Opnieuw kenmerk Text wijzigen
```

```
? MijnFormulier2.Text                    && Geeft "Heel mooi!" als resultaat
? MijnFormulier.Text                     && Geeft "Heel mooi!" als resultaat
```

In Afbeelding 10.5 wordt het resultaat van de voorgaande code weergegeven.

Afbeelding 10.5 MijnFormulier en MijnFormulier2 verwijzen naar hetzelfde formulierobject

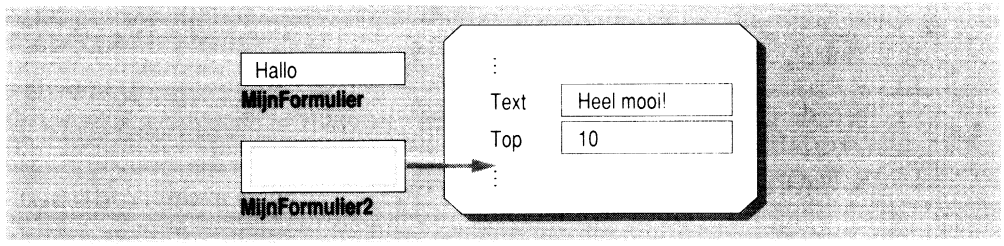


Vervolgens wijst u MijnFormulier toe aan een tekenreeks:

```
MijnFormulier = "Hallo"                  && MijnFormulier is nu variabele van type Teken
? MijnFormulier2.Text                    && Geeft nog steeds "Heel mooi!" als resultaat
```

In Afbeelding 10.6 wordt het resultaat van de voorgaande code weergegeven.

Afbeelding 10.6 MijnFormulier2 verwijst naar een formulier, MijnFormulier is een tekenvariabele



In deze voorbeelden zijn twee belangrijke punten over objectverwijzingsvariabelen duidelijk gemaakt:

- Meerdere objectverwijzingsvariabelen kunnen naar hetzelfde object verwijzen
- Als u de kenmerken van een object wijzigt, worden deze wijzigingen weergegeven door alle verwijzingen naar het object

Objecten in andere objecten maken

Als u een object in een ander object wilt maken, zoals een knop in een formulier, geeft u een verwijzing naar het hoofdobject op met het commando DEFINE of de operator NEW. Hiertoe gaat u als volgt te werk:

- Gebruik het commando DEFINE met de optie OF <verwijzing naar hoofdobject>.

```
DEFINE FORM MijnFormulier                && Hoofdobject maken
DEFINE PUSHBUTTON MijnKnop OF MijnFormulier   && Subobject maken
```

Met DEFINE maakt u maar één verwijzing naar het subobject, namelijk de verwijzing die is gemaakt als lid van het hoofdobject (zie Afbeelding 9.6). In de meeste gevallen is dit de enige verwijzing die u nodig hebt.

- Gebruik de operator NEW en geef een verwijzing naar het hoofdobject door als een parameter.

```
MijnFormulier = NEW FORM()                                  && Hoofdobject maken
MijnFormulier.MijnKnop = NEW PUSHBUTTON(MijnFormulier)      && Verwijzing naar hoofdobject
                                                            && doorgeven als parameter
```

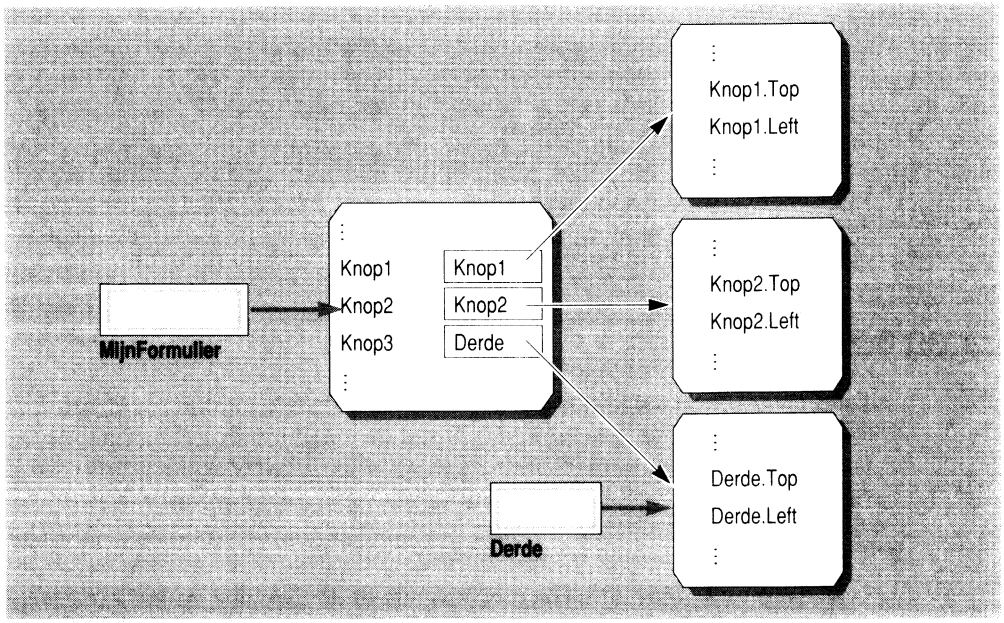
Anders dan met DEFINE kunt u met de operator NEW *twee* verwijzingen naar het subobject maken: een verwijzing die een lid is van het hoofdobject (hetzelfde als bij DEFINE) en een verwijzing die u opgeeft in een toewijzingsopdracht. In de volgende code worden Knop1 en Knop2 gemaakt met één verwijzing en wordt Knop3 gemaakt met twee verwijzingen:

```
DEFINE FORM MijnFormulier                               && Hoofdobject maken
DEFINE PUSHBUTTON Knop1 OF MijnFormulier                && MijnFormulier.Knop1
                                                        && verwijst naar subobject
MijnFormulier.Knop2 = NEW PUSHBUTTON(MijnFormulier)     && MijnFormulier.Knop2
                                                        && verwijst naar subobject
Derde= NEW PUSHBUTTON(MijnFormulier,"Knop3")            && Knop3 en
                                                        && MijnFormulier.Derde verwijzen
                                                        && beide naar het subobject
```

Vervolgens kunt u de eerste en tweede knop benaderen met respectievelijk `MijnFormulier.Knop1.<lidnaam>` of `MijnFormulier.Knop2.<lidnaam>`. U kunt de derde knop benaderen met een van de volgende verwijzingen: `MijnFormulier.Knop3.<lidnaam>` of `Derde.<lidnaam>`.

In Afbeelding 10.7 wordt het resultaat van de voorgaande code weergegeven.

Afbeelding 10.7 Verwijzingen naar subobjecten



Het is handig als u een extra verwijzing naar een subobject maakt, zeker als dit object zich bevindt op verschillende objectniveau's. Een voorbeeld hiervan is een menu-opdracht in een vervolgmenu in een afrolmenu in een menubalk. In plaats van het hele pad van objectverwijzingen op te geven, kunt u volstaan met een extra verwijzing naar de menu-opdracht. (In Hoofdstuk 16 wordt beschreven hoe u menu's maakt voor formulieren.)

Verwijzen naar hoofd- en subobjecten

In de meeste gevallen verwijst u naar de kenmerken van een object met de methoden die bij het object behoren. De methoden kunnen verwijzen naar de eigenschappen door middel van het sleutelwoord *this*, zoals eerder is beschreven. Soms moet een methode echter verwijzen naar de kenmerken van een *ander* object.

Zoals eerder in dit hoofdstuk is beschreven, kunt u naar leden van objecten in andere objecten verwijzen door een pad van verwijzingen samen te stellen met behulp van de stipnotatie. Hier volgt een eenvoudig voorbeeld:

```
DEFINE FORM MijnFormulier                    && Hoofdobject maken
DEFINE PUSHBUTTON MijnKnop OF MijnFormulier  && Knop in formulier maken
MijnFormulier.MijnKnop.Top = 2               && Verwijzen naar kenmerk Top van MijnKnop
                                             && in MijnFormulier
```

U kunt ook naar hoofdobjecten verwijzen met behulp van het kenmerk Parent van subobjecten. Alle ingebouwde objecten van de gebruikersinterface, behalve formulieren, hebben een kenmerk Parent dat een verwijzing bevat naar het hoofdobject. In de meeste gevallen verwijst het kenmerk Parent naar een formulier. Een uitzondering

hierop vormen menu-objecten, die andere menu-objecten kunnen bevatten. In dit voorbeeld wordt een actie-afhandelingsroutine OnClick voor MijnKnop gemaakt die via het kenmerk Parent verwijst naar een kenmerk van MijnFormulier:

```
MijnFormulier.MijnKnop.OnClick = Hoi
PROCEDURE Hoi
   this.Parent.Text = "Hoi!"
RETURN
```

Voor de meeste formulierobjecten is `this.Parent` hetzelfde als `form`. Doorgaans gebruikt u `this.Parent` voor actie-afhandelingsroutines OnClick van subobjecten (menu-opdrachten) als u wilt verwijzen naar het hoofdobject (hoofdmenu).

Eigen objecten maken

Tot dusver hebt u geleerd hoe u ingebouwde gebruikersinterface-objecten voor formulieren maakt en ermee werkt. Naast de ingebouwde gebruikersinterface-objecten kunt u met behulp van de klasse Object uw eigen objecten maken voor willekeurige doeleinden. Een object van de klasse Object heeft geen ingebouwde kenmerken of methoden, maar is een lege container. U maakt een object door de container te maken en de gewenste leden eraan toe te voegen.

Evenals array's gebruikt u eigen objecten voor de tijdelijke opslag van groepen gerelateerde variabelen. (Zie "Werken met array's als objecten" verderop in dit hoofdstuk.) Naarmate u meer leert over object-georiënteerd programmeren, zult u eigen objecten meer en gevarieerder gebruiken. Hier zijn een paar voorbeelden:

- Maak een object Mensen voor de tijdelijke opslag van namen en adressen van klanten, relaties of werknemers.

```
oMensen = NEW Object()              && Object Mensen maken
oMensen.Voornaam = "Jan"
oMensen.Achternaam = "Janssen"
oMensen.Adres = "Waterstraat 23"
oMensen.Plaats = "Haarlem"
oMensen.Regio = "NH"
```

- Maak een object Parameters voor de opslag van parameters die worden doorgegeven aan een functie of procedure. In het derde voorbeeld van de volgende sectie ziet u hoe dat in zijn werk gaat.

Objecten van de klasse Objecten maken is een snelle en eenvoudige manier om te experimenteren met objecten. Als u een object opnieuw wilt kunnen gebruiken, wat diverse voordelen biedt, kunt u een eigen klasse definiëren die dient als een sjabloon voor het maken van meerdere exemplaren van een object. In Hoofdstuk 11 wordt beschreven hoe u klassen definieert.

Objectverwijzingen doorgeven als parameters en resultaatwaarden

Evenals bij andere geheugenvariabelen, kunt u objectverwijzingen als parameters doorgeven aan procedures en functies en kunt u de objectverwijzingen teruggeven als resultaatwaarden. U geeft objectverwijzingen als volgt door als parameter:

- Schrijf een subroutine die een objectverwijzing als een parameter ontvangt en er een handeling mee uitvoert. In dit voorbeeld wordt een verwijzing naar een gebruikersinterface-object ontvangen en in een formulier gecentreerd:

```
PROCEDURE ObjectCentreren
   PARAMETERS oObject                          && Object dat moet worden gecentreerd
   LOCAL nFormWidth
   nFormWidth = oObject.Parent.Width           && Formulier benaderen via kenmerk
                                               && Parent
   oObject.Left = (nFormWidth - oObject.Width) / 2
RETURN
```

- Definieer een functie die een exemplaar van een klasse maakt en die een verwijzing naar het nieuwe object als resultaat geeft. In dit voorbeeld wordt een verwijzing naar een knop als resultaat gegeven:

```
Function NieuweKnop (KnopTekst,FormRef,Top,Left)
   LOCAL NieuweKnop
   NieuweKnop = NEW PUSHBUTTON(FormRef)
   NieuweKnop.Top = Top
   NieuweKnop.Left = Left
   NieuweKnop.Text = KnopTekst
RETURN NieuweKnop

MijnFormulier = NEW FORM()                                  && Formulier maken
MijnFormulier.B = NieuweKnop("Hallo",MijnFormulier,2,2)     && UDF aanroepen om
                                                            && knop toe te voegen
? MijnFormulier.B.Text                                      && Geeft Hallo als resultaat
```

- Meerdere parameters onderbrengen in één object, zodat u maar één parameter (de objectverwijzing) hoeft door te geven in plaats van vele.

```
Function NieuweKnop(oRef)                   && Eén parameter definiëren
   LOCAL NieuweKnop
   NieuweKnop = NEW PUSHBUTTON(oRef.FormRef)
   NieuweKnop.Top = oRef.Top
   NieuweKnop.Left = oRef.Left
   NieuweKnop.Text = oRef.KnopTekst
RETURN NieuweKnop

oParams = NEW Object()                      && Leeg object maken
oParams.KnopTekst = "Hallo"                 && en opvullen met kenmerken
oParams.FormRef = NEW FORM()                && die anders als parameters worden
oParams.Top = 2                             && doorgegeven aan functie
oParams.Left = 2

MijnFormulier.B = NieuweKnop(oParams)       && UDF aanroepen om knop toe
                                            && te voegen
? MijnFormulier.B.Text                      && Geeft Hallo als resultaat
```

Objecten vrijgeven

Bij traditionele geheugenvariabelen wordt de levensduur van de variabele bepaald door het bereik, zoals LOCAL of PRIVATE. De levensduur van een variabele van het type LOCAL is bijvoorbeeld beperkt tot de subroutine waarin de variabele wordt gedefinieerd (in Hoofdstuk 5 wordt het bereik van geheugenvariabelen beschreven.)

Een object heeft dezelfde levensduur als een variabele van het type PUBLIC, met dien verstande dat een object automatisch wordt vrijgegeven wanneer alle verwijzingsvariabelen die naar het object verwijzen, worden vrijgegeven.

In het volgende voorbeeld wordt dit toegelicht:

```
MijnObj = NEW Object()          && Object heeft één verwijzing: MijnObj
MijnObj2 = MijnObj              && Object heeft twee verwijzingen: MijnObj en MijnObj2
RELEASE MijnObj                 && Variabele MijnObj wordt vrijgegeven, object bestaat nog
RELEASE MijnObj2                && MijnObj2 wordt vrijgegeven en het object eveneens
```

Evenals traditionele geheugenvariabelen kunnen objecten te allen tijde expliciet worden vrijgeven, en wel op de volgende manieren:

- Met de methode Release van het object. Alle gebruikersinterface-objecten, zoals invoervakken en knoppen, hebben een ingebouwde methode Release.
- Met het commando RELEASE OBJECT. Met RELEASE OBJECT geeft u het object vrij waarnaar wordt verwezen door een objectverwijzingsvariabele. De variabele wordt echter niet vrijgegeven.

Belangrijk Let op het verschil tussen RELEASE OBJECT in het volgende voorbeeld en RELEASE in het voorgaande voorbeeld:

```
MijnObj = NEW Object()          && Object heeft één verwijzing: MijnObj
MijnObj.Naam = "Jan"            && Kenmerk toevoegen
MijnObj2 = MijnObj              && Object heeft twee verwijzingen: MijnObj en MijnObj2
RELEASE OBJECT MijnObj          && Object waarnaar wordt verwezen door MijnObj, wordt
                                && vrijgegeven; MijnObj bestaat nog
? MijnObj                       && Geeft "Object" als resultaat, hoewel dit nergens naar
                                && verwijst
? EMPTY(MijnObj)                && Geeft .T. als resultaat
? MijnObj.Naam                  && Geeft een foutmelding: Poging een vrijgegeven object te
                                && benaderen
RELEASE MijnObj,MijnObj2        && MijnObj en MijnObj2 worden vrijgegeven
```

Als u gebruikersinterface-objecten maakt, zoals formulieren, knoppen en invoervakken, wordt automatisch een interne verwijzing gemaakt naast de verwijzingen die u zelf maakt. Hierdoor bent u ervan verzekerd dat er altijd minimaal één verwijzing bestaat voor elk object. Het gevolg is dat u een gebruikersinterface-object alleen kunt vrijgeven met RELEASE OBJECT of met de methode Release van het object.

```
MijnFormulier = NEW FORM()              && Formulier heeft twee verwijzingen:
                                        && MijnFormulier en een interne verwijzing
DEFINE TEXT MijnTekst OF MijnFormulier  && Object heeft twee verwijzingen: MijnTekst
                                        && en een interne verwijzing
DEFINE ENTRYFIELD MijnInvoer OF MijnFormulier
RELEASE MijnFormulier                   && Formulier heeft nog steeds één verwijzing:
```

```
                                    && de interne
RELEASE MijnTekst                   && Object wordt niet vrijgegeven
```

Als u een object vrijgeeft dat andere objecten bevat, zoals een formulier met knoppen, worden alle bijbehorende subobjecten eveneens vrijgegeven.

Werken met array's als objecten

In Hoofdstuk 9 zijn objecten geïntroduceerd als verzamelingen geheugenvariabelen die vergelijkbaar zijn met array's. In feite *zijn* array's objecten in dBASE. Als u een array maakt, maakt u een object dat is gebaseerd op de ingebouwde klasse Array. U kunt array's op dezelfde manier gebruiken als in dBASE IV. (In Hoofdstuk 5 wordt beschreven hoe u array's kunt gebruiken op de manier die u gewend was in dBASE IV.) Wanneer u array's als objecten gebruikt, beschikt u over de volgende nieuwe mogelijkheden in dBASE:

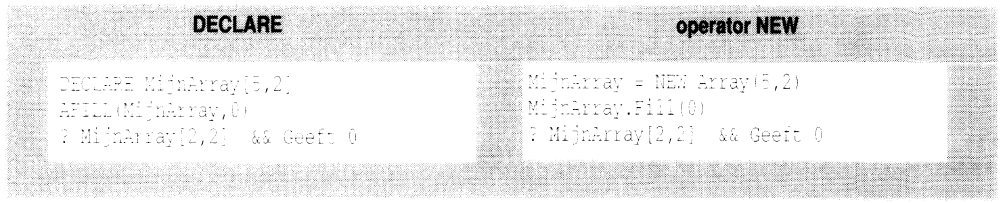
- *Het formaat ophalen met het kenmerk Size.* Array's hebben een kenmerk Size waarmee het aantal elementen in de array wordt weergegeven. Bij 1-dimensionale array's kunt u het kenmerk Size wijzigen als u het formaat van de array wilt aanpassen. Bij 2-, 3- of meer-dimensionale array's is het kenmerk Size alleen-lezen. Gebruik de methode Resize() als u het formaat van array's met meer dan 1 dimensie wilt wijzigen.
- *Ingebouwde methoden aanroepen* als u elementen wilt toevoegen of verwijderen, als u elementen wilt opvullen met waarden, als u het formaat van de array wilt wijzigen, enzovoort. Alle array-functies, zoals AADD() of AFIELDS(), hebben corresponderende methoden. De methoden hebben dezelfde naam als de functies, maar zonder de beginletter "A".
- *Uw eigen methoden toevoegen.* Evenals aan een object, kunt u een kenmerk toevoegen aan een array die verwijst naar een eigen subroutine.
- *Eigen arrayklassen definiëren* als u eigen array's wilt maken met standaardwaarden die u opgeeft of met eigen methoden die u definieert.

Als u een array wilt maken, kunt u het commando DECLARE gebruiken of de operator NEW. Met DECLARE moet u de dimensies van de array opgeven. Met de operator NEW kunt u de dimensies doorgeven als parameters of kunt u geen parameters opgeven. In het laatste geval kunt u elementen later toevoegen met een van de methoden voor array's.

Belangrijk Anders dan bij andere ingebouwde klassen, kunt u DEFINE niet gebruiken om een array-object te maken.

In Afbeelding 10.8 ziet u equivalente manieren om een array te maken en initialiseren:

Afbeelding 10.8 Equivalente manieren om array's te maken en initialiseren



U kunt de methoden van een array gebruiken ongeacht de manier waarop u de array definieert.

```
DECLARE EenArray[6,10]                  && Array declareren met 6 rijen en 10 kolommen
EenArray.Dir()                          && Array opvullen met huidige directorylijst
```

In Tabel 10.1 wordt een overzicht gegeven van de methoden die beschikbaar zijn voor array's:

Tabel 10.1 Overzicht van methoden voor klasse Array

Methode	Beschrijving
Add()	Voegt een element toe
Delete()	Verwijdert een element, kolom of rij
Dir()	Vult de array op met de huidige directorylijst.
Element()	Geeft het elementnummer van een opgegeven indexteken van een element (rij en kolom) als resultaat
Fields()	Vult de array op met de structuur van de huidige tabel
Fill()	Voegt een opgegeven waarde in een of meer elementen in
Grow()	Voegt een element, kolom of rij toe
Insert()	Voegt de waarde False (.F.) in een opgegeven element in
Resize()	Vergroot of verkleint de array
Scan()	Zoekt in de array naar een opgegeven uitdrukking
Sort()	Sorteert elementen in 1- en 2-dimensionale array's
Subscript()	Geeft het indexteken (rij en kolom) van een opgegeven elementnummer als resultaat

Grow(), Size

In dit voorbeeld wordt een array gemaakt met evenveel elementen als er records zij in de tabel KLANTEN.DBF. Vervolgens wordt de array opgevuld met waarden uit de tabel, wordt Grow() gebruikt om een tweede kolom toe te voegen, wordt de tweede kolom opgevuld met waarden uit de tabel en worden de resultaten weergegeven.

```
USE Klanten.DBF
ObjArr=NEW ARRAY(RECCOUNT())            && Array-object initialiseren
GO TOP
SCAN                                    && 1-dimensionale array opvullen met waarden
  ObjArr[RECNO()]=Klanten->Naam         && uit veld Naam van KLANTEN.DBF
ENDSCAN

ObjArr.GROW(1)                          && Met GROW() tweede kolom toevoegen
```

```
GO TOP                                    && aan bestaande array, en Verkotnu-gegevens
SCAN                                      && invoeren in nieuwe tweede kolom
   ObjArr[RECNO(),2]=Klanten->Verkotnu
ENDSCAN

FOR i = 1 TO ObjArr.Size                  && Verwijzing maken naar kenmerk Size
   ? ObjArr[nCount,1], ObjArr[nCount,2]   && Inhoud van 2-dimensionale array weergeven
NEXT
```

Add()

In dit voorbeeld wordt het voorgaande voorbeeld vereenvoudigd door de methode Add() te gebruiken om de array op te vullen.

```
USE KLANTEN.DBF
ObjArr=NEW ARRAY()
SCAN
   ObjArr.Add(Field)
ENDSCAN

FOR i = 1 TO ObjArr.Size
   ? ObjArr[i]
NEXT
```

Scan(), Subscript()

In dit voorbeeld ziet u een andere manier om een array op te vullen met waarden uit een tabel. Na het opvullen wordt met Scan() gezocht naar een reeks in de array en wordt Subscript() gebruikt om de lokatie weer te geven.

```
USE Dieren.DBF
ObjArr=NEW ARRAY(RECCOUNT(),3)                    && Array-object initialiseren
COPY TO ARRAY ObjArr FIELDS Naam, Gebied, Gewicht  && Array opvullen met waarden uit tabel

String = "Papegaai"
aElement = ObjArr.Scan(String)                    && In array zoeken naar reeks
aRij = ObjArr.SUBSCRIPT(aElement,1)
aKol = ObjArr.SUBSCRIPT(aElement,2)
? String +" bevindt zich in Rij" + ;
  LTRIM(STR(aRij)) + ", Kolom " + ;
  LTRIM(STR(aKol))
```

Minimale array's maken

U kunt elementen aan array's toevoegen nadat u deze hebt gedefinieerd. Elk element dat u toevoegt, moet echter het volgende, opeenvolgende elementnummer zijn. Als een array bijvoorbeeld 99 elementen heeft, voegt u met de methode Add() element 100 toe. Een *minimale array* is een array met een variabel aantal niet-aaneengesloten elementen. Een minimale array kan bijvoorbeeld drie elementen bevatten met de nummers 1, 50 en 300.

Met minimale array's kunt u geheugen efficiënter gebruiken, omdat deze alleen het aantal elementen bevatten dat u nodig hebt. Bovendien kan de indexlokatie van een element een betekenisvol gegeven zijn in plaats van alleen een lokatienummer.

U maakt minimale array's door objecten van de klasse Objects te maken. U voegt elementen toe door een numerieke waarde op te geven met de index-operator. In de volgende code wordt een minimale array gemaakt en worden drie elementen met niet-aaneengesloten indexlokaties toegevoegd.

```
oSparse = NEW Object()
oSparse[1] = "Eerste element"
oSparse[50] = "Tweede element"
oSparse[300] = "Derde element"
```

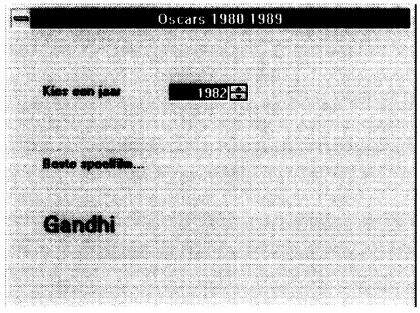
Stel dat u een array wilt maken voor het opslaan van de winnaars van een Oscar voor de beste film in de periode 1980-1989. Met een minimale array kunt u het jaar gebruiken als de indexlokatie zonder dat u niet-gebruikte elementen maakt. In het volgende voorbeeld wordt een minimale array met de naam BesteFilm gemaakt, worden de Oscar-winnaars voor elk jaar in de periode 1980-1989 opgeslagen en wordt de array gekoppeld aan een ringveld in een formulier.

```
BesteFilm= NEW OBJECT()
BesteFilm[1980] = "Ordinary People"
BesteFilm[1981] = "Chariots of Fire"
BesteFilm[1982] = "Gandhi"
BesteFilm[1983] = "Terms of Endearment"
BesteFilm[1984] = "Amadeus"
BesteFilm[1985] = "Out of Africa"
BesteFilm[1986] = "Platoon"
BesteFilm[1987] = "The Last Emperor"
BesteFilm[1988] = "Rain Man"
BesteFilm[1989] = "Driving Miss Daisy"

BestForm = NEW FilmForm()
BestForm.ReadModal()

CLASS FilmForm OF FORM
      this.MDI = .F.
      this.Text = "Oscars 1980-1989"
      DEFINE TEXT Yprompt OF this ;
            PROPERTY ;
            top    3, ;
            left   5, ;
            width 20, ;
            text  "Kies een jaar"
      DEFINE SPINBOX Jaren OF this ;
            PROPERTY ;
            top      3, ;
            left     25, ;
            rangemin 1980, ;
            rangemax 1989, ;
            value    1980, ;
            onChange {;form.Film2.Text = BesteFilm[this.Value]}
```

```
        DEFINE TEXT Film1 OF this ;
               PROPERTY ;
               top    7, ;
               left   5, ;
               width 20, ;
               text  "Beste speelfilm..."
        DEFINE TEXT Film2 OF this ;
               PROPERTY ;
               top      10, ;
               left      5, ;
               width    40, ;
               height    2, ;
               fontSize 15, ;
               text BesteFilm[form.Jaren.Value]
ENDCLASS
```

In Afbeelding 10.9 wordt het resultaat van de voorgaande code weergegeven.

Afbeelding 10.9 BESTFILM.PRG: een minimale array



Hoofdstuk

11

Werken met klassen

In Hoofdstuk 9 wordt beschreven hoe u objecten maakt op basis van ingebouwde klassen. Tot de krachtigste functies van het dBASE-objectmodel behoort de mogelijkheid om eigen klassen te definiëren voor het maken van objecten, en eigen klassen te definiëren die zijn gebaseerd op andere klassen. In dit hoofdstuk worden de grondbeginselen van het werken met klassen uitgelegd, klassehiërarchieën geïntroduceerd en technieken beschreven voor het maken en gebruiken van klassen die zijn gebaseerd op andere klassen.

Klassen definiëren

U definieert een nieuwe klasse met het commando CLASS...ENDCLASS. Zie *Commando's en functies* voor een volledige beschrijving van CLASS...ENDCLASS. Hier volgt een samenvatting van de syntaxis:

```
CLASS <naam subklasse> [(<parameters>)] [OF <naam hoofdklasse> (<parameters>)]
   [PARAMETERS <parameters>]
   <constructor-code>
   <definities van methoden>
ENDCLASS
```

De kern van een klassedefinitie bestaat uit de volgende delen:

- *Constructor-code*, die wordt uitgevoerd wanneer een object van de desbetreffende klasse wordt gemaakt. Constructor-code is de code waarmee de objectleden feitelijk worden gemaakt. Deze code kan willekeurige dBASE-commando's bevatten. In de meeste gevallen bevat deze code alleen toewijzingsopdrachten voor kenmerken en methoden. Constructor-code wordt gebruikt om de objectkenmerken te maken en de bijbehorende beginwaarden in te stellen.
- *Definities van methoden*, zoals procedures of functies. Een methode bevat code die wordt uitgevoerd nadat het object is gemaakt. Deze subroutines worden gebruikt om bewerkingen uit te voeren op de kenmerken.

De volgende code bevat een eenvoudige klassedefinitie:

```
CLASS Artikel                    && Klasse benoemen.
   This.Artikel = "Tas"          && Deze constructor-code initialiseert kenmerken.
   This.Maat = 12                && Constructor-code loopt door tot eerste definitie
   This.Prijs = 16.95            && van methode of tot een opdracht ENDCLASS,
   This.Aant = 4                 && afhankelijk van wat het eerst wordt aangetroffen.
   Function Reken                && Methode definiëren.
      LOCAL X
      X=This.Prijs*This.Aant
   Return X
ENDCLASS                         && Klassedefinitie beëindigen
```

Als u een object van deze klasse met de naam MijnArtikelen wilt maken, gebruikt u de volgende code:

```
DEFINE Artikel MijnArtikelen
```

of

```
MijnArtikelen = NEW Artikel()
```

In het volgende voorbeeld wordt een klasse met twee kenmerken gedefinieerd, wordt een object van die klasse gemaakt en wordt een query uitgevoerd op de waarde van de kenmerken:

```
CLASS Getallen
   this.Tien = 10
   this.Twintig = 20
ENDCLASS

X = NEW Getallen()     && Maakt een nieuw object Getallen
? X.Tien               && Resulteert in 10
? X.Twintig            && Resulteert in 20
```

Verwijzen naar leden in een klassedefinitie

Als u wilt verwijzen naar leden in een klassedefinitie, gebruikt u *this* of *form* als voorvoegsel voor de naam van het kenmerk.

- *This* verwijst naar het object dat door de klasse wordt gemaakt. Een toewijzingsopdracht die begint met *this*, definieert een kenmerk voor objecten die door de klasse worden gemaakt.
- *Form* verwijst naar het formulier dat het door de klasse gemaakte object bevat. Gebruik *form* in klassedefinities voor stuurelementen, zoals invoervakken of knoppen, als u wilt verwijzen naar het formulier dat het door de klasse gemaakte object bevat.

In de volgende klassedefinitie wordt *this* gebruikt om de kenmerken Getal en KwadraatGetal te initialiseren en om te verwijzen naar Getal in de toewijzingsopdracht voor KwadraatGetal.

```
CLASS Kwadraat
```

```
   this.Getal = 0
   this.KwadraatGetal = this.Getal * this.Getal
ENDCLASS
```

U kunt geheugenvariabelen definiëren in constructor-code (dat wil zeggen, variabelen die niet worden gedefinieerd of waarnaar niet wordt verwezen als kenmerken) als u waarden tijdelijk wilt opslaan terwijl u het object maakt. De regels voor bereiken die gelden voor de klasse, zijn van toepassing op geheugenvariabelen in constructor-code. U kunt variabelen van de typen LOCAL, PRIVATE en PUBLIC definiëren. In constructor-code kunt u echter geen variabelen van het type STATIC definiëren.

In de volgende code wordt een voorbeeld gegeven van het gebruik van geheugenvariabelen in constructor-code. Class CentreerKnop definieert een knop die automatisch gecentreerd in het formulier wordt weergegeven. Met behulp van de variabele *nFormulierBreedte* wordt het midden berekend.

```
CLASS CentreerKnop OF PUSHBUTTON(f)
   LOCAL nFormulierBreedte
   nFormulierBreedte = this.Parent.Width       && Formulier benaderen via kenmerk Parent
   this.Left = (nFormulierBreedte - this.Width) / 2
RETURN
```

Methoden verbinden met een klasse

Een methode is aanvankelijk geen methode maar een subroutine, zoals een procedure, functie of codeblok. De subroutine wordt een methode als u deze verbindt met een klasse. U verbindt de subroutine met een klasse door de subroutine toe te wijzen aan een variabele van het type Functie-aanwijzer of Codeblok die lid van een klasse is.

U kunt methoden op de drie volgende manieren verbinden met klassen:

- Definieer een functie of procedure *in* de klassedefinitie. Voor elke subroutine die u in de klasse definieert, maakt dBASE automatisch een variabele van het type Functie-aanwijzer met dezelfde naam in de klasse. Door deze techniek wordt de subroutine in de klasse ingesloten, zodat een andere klasse een methode met dezelfde naam kan bevatten. Met Formulierontwerp definieert u een methode op deze wijze als u de Procedure-editor gebruikt om een subroutine te verbinden met een actiekenmerk.

```
CLASS MijnObj
   this.text = "Dit is mijn object"   && Enkele kenmerken toewijzen
   this.top = 10
   this.left = 5

   PROCEDURE Biep                     && Deze proceduredefinities maken methoden.
      ? chr(7)                        && Als u een object op basis van deze klasse maakt,
   Return .T.                         && zijn de leden Biep en TotZiens functie-
   PROCEDURE TotZiens                 && aanwijzers die naar de procedures verwijzen
      ? "Dag wereld!"
      RELEASE OBJECT this
   RETURN .T.
ENDCLASS
```

- Definieer een functie of procedure *buiten* de klassedefinitie en wijs een kenmerk in de klasse toe aan de naam van de subroutine. Met behulp van deze techniek kunt u de

subroutine in een procedurebestand plaatsen waar de subroutine kan dienen als methode voor meerdere klassen.

```
CLASS MijnObj
   this.text = "Dit is mijn object"      && Enkele kenmerken toewijzen
   this.top = 10
   this.left = 5

   this.Biep = Biep                      && Procedurenamen toewijzen
   this.TotZiens = TotZiens              && om methoden te maken
ENDCLASS

PROCEDURE Biep                           && Biep-geluid genereren
    ? chr(7)
Return .T.

PROCEDURE TotZiens                       && Reeks "Dag wereld!" weergeven
   ? "Dag wereld!"
   RELEASE OBJECT this
RETURN .T.
```

- Wijs een codeblok toe aan een variabele in een klassedefinitie. Dit is de eenvoudigste manier om een methode te definiëren, met name wanneer de code voor de methode maar weinig commando's omvat.

```
CLASS MijnObj
   this.text = "Dit is mijn object"   && Enkele kenmerken toewijzen
   this.top = 10
   this.left = 5

   this.Biep = {;?chr(7)}             && Twee methoden toewijzen met codeblokken
   this.TotZiens = {;? "Dag wereld!";RELEASE OBJECT this}
ENDCLASS
```

In het voorbeeld met Kwadraat op bladzijde 152-153 wordt de waarde van het kenmerk KwadraatGetal berekend wanneer u het object maakt. Als u KwadraatGetal definieert als een methode in plaats van een kenmerk, kunt u KwadraatGetal op elk gewenst moment berekenen. In het volgende voorbeeld wordt dit toegelicht:

```
X = NEW Kwadraat2()           && Nieuw object Kwadraat2 maken
X.Getal = 5                   && Waarde van Getal wijzigen
? X.KwadraatGetal()           && Resulteert in 25
X.Getal = 6                   && Waarde van Getal nogmaals wijzigen
? X.KwadraatGetal()           && Resulteert in 36

CLASS Kwadraat2
   this.Getal = 0
   FUNCTION KwadraatGetal
   RETURN this.Getal * this.Getal
ENDCLASS
```

Een subroutine kan als methode dienen voor meerdere objecten. In het volgende voorbeeld wordt dit toegelicht:

```
O = NEW Object()
```

```
O.x = 10
O.éénOpt = FuncEénOpt
? O.éénOpt()              && Resulteert in '11'
? O.x                     && Resulteert in '11'
Y = NEW Object()
Y.x =30
Y.z = FuncEénOpt
? Y.z()                   && Resulteert in '31'

FUNCTION FuncEénOpt
   this.x = this.x + 1
RETURN this.x
```

Als u dezelfde naam toewijst aan een methode en een kenmerk, wordt de naam van het kenmerk gebruikt in uitdrukkingen. In dit voorbeeld wordt de naam Tien toegewezen aan zowel een methode als een kenmerk. Let op de resultaatwaarden.

```
FUNCTION Tien             && Methode met de naam Tien definiëren
RETURN 10

Tien = 10                 && Variabele met de naam Tien definiëren
pFunc = Tien              && Waarde 10 toewijzen aan pFunc
? pFunc()                 && FOUT: niet-overeenkomend gegevenstype
```

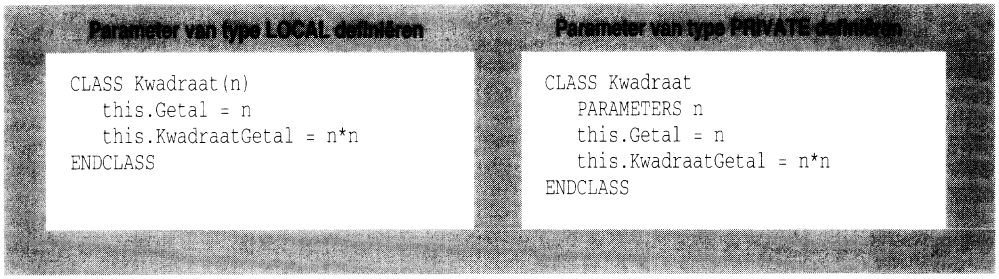
Parameters voor een klasse definiëren

U kunt in een klassedefinitie parameters definiëren die moeten worden doorgegeven aan de constructor-code wanneer u het object maakt. U definieert klasseparameters op dezelfde wijze als u parameters voor functies definieert. U kunt klasseparameters op de volgende twee manieren definiëren:

- Plaats de parameters tussen haakjes aan het einde van de klassenaam. Deze methode verdient de voorkeur omdat het bereik van parameters die op deze manier worden doorgegeven, LOCAL is voor de klasse.
- Gebruik een opdracht PARAMETERS als eerste regel van de constructor-code. dBASE-programmeurs zijn wellicht meer vertrouwd met deze wijze om parameters te definiëren. In dit geval is het bereik van de parameters echter PRIVATE voor de klasse. In Hoofdstuk 5 wordt beschreven waarom LOCAL meer geschikt is als bereik voor parameters.

De volgende twee klassedefinities zijn identiek, met dien verstande dat in de ene definitie een parameter met haakjes wordt gedefinieerd en in de andere definitie met PARAMETERS.

Afbeelding 11.1 Parameters definiëren met bereik LOCAL of PRIVATE



In de volgende code worden op basis van een van deze klassen objecten gemaakt:

```
X = NEW Kwadraat(10)        && Nieuw object Kwadraat maken
? X.KwadraatGetal           && Resulteert in 100
Y = NEW Kwadraat(5)         && Nieuw object Vierkant maken
? Y.KwadraatGetal           && Resulteert in 25
```

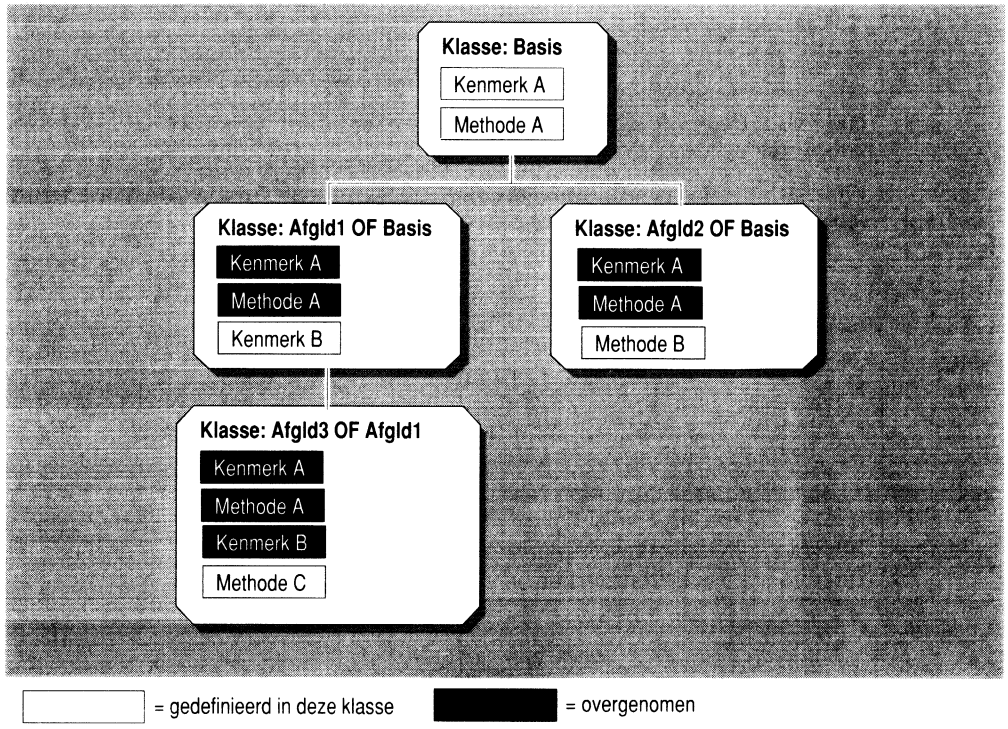
Hiërarchieën van klassen

Een klasse die is gebaseerd op een andere klasse, wordt een *subklasse genoemd*. Een subklasse neemt de leden over van de klasse waarop de subklasse is gebaseerd, zodat u leden kunt toevoegen aan de subklasse of de bestaande leden kunt wijzigen. Op deze wijze kunt u bestaande klassen, zoals de ingebouwde klasse Form of een klasse die u zelf definieert, gebruiken om nieuwe, soortgelijke klassen te maken. Telkens wanneer u een formulier maakt met Formulierontwerp, wordt een nieuwe klasse gemaakt op basis van de ingebouwde klasse Form.

Afgeleide subklassen bieden het meeste voordeel als u een hiërarchie van klassen samenstelt, waarbij de functionaliteit van elke subklasse voor het grootste deel wordt overgenomen van een hoofdklasse. In een subklasse wordt de functionaliteit gewijzigd door overgenomen leden te vervangen of nieuwe leden toe te voegen. U kunt code dan gemakkelijk bijwerken en de code is in hoge mate opnieuw bruikbaar.

In Afbeelding 11.2 ziet u een voorbeeld van een klassehiërarchie.

Afbeelding 11.2 Een voorbeeld van klassehiërarchie



In deze afbeelding geldt:

- Basis is een hoofdklasse voor Afgld1 en Afgld2.
- Afgld1 en Afgld2 zijn subklassen van Basis.
- Afgld1 is een hoofdklasse voor Afgld3.

Overname van kenmerken en methoden tussen klassen is iets anders als de situatie waarbij een object andere objecten bevat (een object *container* is van andere objecten). In Hoofdstuk 10 wordt het begrip container in verband met objecten uitgelegd.) Als een object een ander object bevat, bestaat er een exemplaar van elk subobject dat hoort bij het hoofdobject van het subobject. Het subobject is een *onderdeel van* het hoofdobject. Als u daarentegen een subklasse afleidt van een hoofdklasse, heeft de subklasse alleen een aantal kenmerken gemeen met de hoofdklasse. De subklasse is een *type van* de bijbehorende hoofdklasse.

Een klasse definiëren die is gebaseerd op een andere klasse

Als u een klasse wilt definiëren die is gebaseerd op een andere klasse, gebruikt u de optie OF bij het commando CLASS...ENDCLASS. Met deze optie geeft u de hoofdklasse op waarvan leden moeten worden overgenomen. De elementaire syntaxis is als volgt:

```
CLASS <naam subklasse> [(<parameters>)] OF <naam hoofdklasse> [(<parameters>)]
   <constructor-code>
   <definities van methoden>
ENDCLASS
```

Als u een nieuw object op basis van een subklasse maakt, wordt eerst de constructor-code voor de hoofdklasse uitgevoerd, gevolgd door de constructor-code voor de subklasse.

Hieronder ziet u een voorbeeld van de werking van subklassen. Eerst worden de definities voor een hoofdklasse en subklasse gegeven:

```
CLASS Basis
   ? "klasse Basis"     && Melding weergeven tijdens constructie
   this.X = 10          && X initialiseren
   this.Y = 20          && Y initialiseren
ENDCLASS

CLASS Sub OF Basis      && Klasse Sub bevat leden X en Y van Basis
   ?? ", Subklasse"     && Melding weergeven tijdens constructie
   this.Z = 100         && Nieuw kenmerk definiëren, alleen voor subklasse
ENDCLASS
```

Vervolgens worden exemplaren van deze klassen gemaakt en wordt een query uitgevoerd op de kenmerken.

```
b = NEW Basis()         && Resulteert in 'klasse Basis'
? b.X                   && Resulteert in 10
? b.Y                   && Resulteert in 20
s= NEW Sub()            && Resulteert in 'klasse Basis, Subklasse'
? s.X                   && Resulteert in 10
? s.Y                   && Resulteert in 20
? s.Z                   && Resulteert in 100
```

Belangrijk Als u een stuurelement maakt op basis van een standaardklasse, zoals Entryfield of Pushbutton, moet u het formulier opgeven waarin het stuurelement wordt geplaatst. In het volgende voorbeeld is de optie `OF MijnFormulier` dan ook vereist.

```
DEFINE ENTRYFIELD MijnInvoer OF MijnFormulier
```

Als u een subklasse definieert die is gebaseerd op een standaardklasse, moet u een parameter voor de klasse definiëren waarin de verwijzing naar het formulier kan worden geplaatst. Het is niet van belang welke naam u aan de parameter geeft.

In het volgende voorbeeld wordt een subklasse gedefinieerd op basis van de standaardklasse Pushbutton. Klasse MijnKnop dient als een standaardklasse voor knoppen en bevat de instellingen die door alle knoppen worden overgenomen die van deze klasse worden afgeleid. Vervolgens wordt BiepKnop (een knop die een signaal genereert) afgeleid van MijnKnop.

```
CLASS MijnKnop(form) OF PUSHBUTTON(form)        && Form-parameters definiëren
   this.text = "Mijn knop"                      && Kenmerken voor alle knoppen
   this.FontName = "Arial"                      && toewijzen
   this.FontBold = .T.
   this.FontWidth = 6
   this.SetWidth = {;this.width = LEN(this.text)+2}
ENDCLASS

CLASS BiepKnop(form) OF MijnKnop(form)          && Form-parameters definiëren
   this.text = "Signaal"                        && Kenmerk Text vervangen
   this.Onclick = {;? CHR(7)}                   && Codeblok toewijzen aan OnClick
   this.SetWidth()                              && Overgenomen methode aanroepen
ENDCLASS
```

Overgenomen leden vervangen

Als u een subklasse afleidt van een andere klasse, kunt u de leden die worden overgenomen van de hoofdklasse, vervangen door de kenmerken opnieuw te initialiseren of door de methoden opnieuw te definiëren.

Hierna volgt een eenvoudig voorbeeld van het vervangen van een overgenomen methode. De klasse Basis bevat twee methoden en een ervan wordt vervangen in de subklasse.

```
CLASS Basis
   FUNCTION Een
   RETURN "Basis 1"
   FUNCTION Twee
   RETURN "Basis 2"
ENDCLASS

CLASS Sub OF Basis
   FUNCTION Twee              && Methode Twee vervangen
   RETURN "Sub 2"
ENDCLASS
```

Vervolgens maakt u exemplaren van deze klassen en roept u de methoden aan.

```
A = NEW Basis()
B = NEW Sub()
? A.Een()                     && Resulteert in 'Basis 1'
? A.Twee()                    && Resulteert in 'Basis 2'
? B.Een()                     && Resulteert in 'Basis 1'
? B.Twee()                    && Resulteert in 'Basis 2'
```

Het volgende voorbeeld is wat meer op de praktijk toegesneden:

```
CLASS BiepKnop(form) OF MijnKnop(form)
   this.text = "Signaal"
   this.Onclick = BiepProc
   this.SetWidth()
   PROCEDURE BiepProc
      ? CHR(7)
   RETURN
ENDCLASS

CLASS BiepVerplaats(form) OF BiepKnop(form)
   this.text = "Signaal en Verplaatsen"    && Kenmerk Text vervangen
   this.SetWidth()
   PROCEDURE OnClick
      BiepKnop::BiepProc()
      this.Left = this.Left+1              && Functionaliteit toevoegen
                                           && aan methode OnClick
   RETURN
ENDCLASS
```

Verwijzen naar methoden vanuit een klasse

Zoals de voorbeelden uit de vorige sectie duidelijk maken, kunt u met behulp van de stipnotatie verwijzen naar de methoden van een object. U kunt ook rechtstreeks vanuit de klasse verwijzen naar methoden. Deze techniek is bijzonder handig als u methoden wilt aanroepen die zijn gedefinieerd op verschillende niveaus in een klassehiërarchie. U kunt bijvoorbeeld een overgenomen methode in een subklasse wijzigen door de methode rechtstreeks vanuit de hoofdklasse aan te roepen en zodoende het gedrag van de methode in de subklasse aan te passen.

Als u vanuit een klasse naar een methode wilt verwijzen, gebruikt u de *bereikoperator* (::). Met de stip-operator verbindt u een lid met het bijbehorende object. Op dezelfde wijze verbindt u met de bereikoperator een methode met de klasse waarin de methode wordt gedefinieerd. De syntaxis is als volgt:

```
<naam klasse> | class | super :: <naam methode>
```

U kunt de naam van een klasse opgeven of de sleutelwoorden *class* of *super gebruiken*. *Class* en *super* zijn voor klassen wat *this* en *form* zijn voor objecten: een manier om in algemene zin naar iets (in dit geval een klasse) te verwijzen zonder een feitelijke naam te gebruiken in de code.

- *Class* verwijst naar de huidige klasse.
- *Super* verwijst naar de hoofdklasse van de huidige klasse.

Evenals *this* en *form* kunt u *class* en *super* alleen gebruiken in een context die aangeeft wat de huidige klasse is. U kunt *class* en *super* als volgt gebruiken:

- In een klassedefinitie, waarbij de huidige klasse de klasse is die wordt gedefinieerd.

  De volgende klasse is gegenereerd met Formulierontwerp. De actie OnClick voor de knop wordt ingesteld op de procedure KNOP1_ONCLICK in de huidige klasse.

```
CLASS Biep OF FORM
   this.EscExit = .T.
   this.Text = "dBASE-formulier"
   this.Width =  65.25
   this.Top =    8.08
   this.Left =   25.00
   this.Height = 34.25

   DEFINE PUSHBUTTON KNOP1 OF this;
      PROPERTY;
         OnClick class::KNOP1_ONCLICK,;      && KNOP1_ONCLICK toewijzen vanuit de
         Text "KNOP1",;                      && huidige klasse
         Width  21.00,;
         Top    12.00,;
         Left   12.00,;
         Height 4.00

   PROCEDURE KNOP1_OnClick
   ? chr(7)
ENDCLASS
```

- In definities van methoden, waarbij de huidige klasse de klasse is van waaruit het object is gemaakt dat de methode bevat.

  In het volgende voorbeeld wordt het vorige voorbeeld met BiepKnop gewijzigd. Procedure OnClick in de klasse BiepVerplaats verwijst naar BiepProc uit de klasse BiepKnop met behulp van *super*:

```
CLASS BiepKnop(form) OF MijnKnop(form)
   this.text = "Signaal"
   this.Onclick = BiepProc
   this.SetWidth()
   PROCEDURE BiepProc
      ? CHR(7)
   RETURN
ENDCLASS


CLASS BiepVerplaats(form) OF BiepKnop(form)
   this.text = "Signaal en Verplaatsen"   && Kenmerk Text vervangen
   this.SetWidth()
   PROCEDURE OnClick
      super::BiepProc()
      this.Left = this.Left+1             && Functionaliteit toevoegen aan methode OnClick
   RETURN
ENDCLASS
```

Met behulp van de bereikoperator kunt u zelfs een methode aanroepen vanuit een klasse als u geen exemplaar van die klasse hebt gemaakt. Om te beginnen moet de klassedefinitie (zoals elke procedure of functie) zich bevinden in het huidige programmabestand of moet de klassedefinitie als een procedurebestand worden geladen met SET PROCEDURE. Vervolgens roept u de methode aan door voor de naam van de methode de klassenaam op te geven met de bereikoperator. Een methode in een klasse neemt feitelijk de naam van de klasse over als deel van de eigen naam.

Hier volgt een eenvoudig voorbeeld:

```
? MijnKlasse::ZegHallo()        && Resulteert in "Hallo!"

CLASS MijnKlasse
   FUNCTION ZegHallo
      ? "Hallo!"
   RETURN .T.
ENDCLASS
```

In het volgende voorbeeld wordt om meer praktische redenen een methode aangeroepen vanuit een klasse. Stel dat u een applicatie schrijft waarmee veel exemplaren van een bepaald formulier worden gemaakt. De applicatie moet weten hoeveel formulieren zijn gemaakt, en bovendien moet de applicatie een verwijzing kunnen maken naar elk van de formulieren.

In de volgende klassedefinitie, VeelForm, resulteert de methode HaalFormTel in het aantal formulieren dat op basis van de klasse is gemaakt. De methode HaalFormVerw resulteert in een verwijzing naar een bepaald formulier. Aangezien u niet weet of en hoeveel formulieren er zijn, worden deze methoden rechtstreeks vanuit de klasse aangeroepen.

```
F = NEW VEELFORM()
F.Open()
? VeelForm::HaalFormTel()                  && Resulteert in aantal gemaakte formulieren
FormEerste = VeelForm::HaalFormVerw(1)     && Resulteert in een verwijzing naar eerst
                                           && gemaakte formulier
* De volgende regel resulteert in een verwijzing naar het laatst gemaakte formulier
FormLaatste = VeelForm::HaalFormVerw(VeelForm::HaalFormTel())

CLASS VeelForm OF FORM
   this.BijOpen = CLASS::FORM_BIJOPEN

   PROCEDURE Form_BijOpen
      PUBLIC XArray                        && xArray slaat een verwijzing naar elk
                                           && gemaakt formulier op
      IF EMPTY(XArray)
         XArray = NEW Array(0)
      ENDIF
      this.Index = XArray.Add(this)
      this.Text = STR(this.Index)
   RETURN

   PROCEDURE HaalFormTel
      IF .NOT. EMPTY(XArray)
         RETURN XArray.Size
      ELSE
         RETURN 0
      ENDIF
   RETURN
```

```
   PROCEDURE HaalFormVerw(i)
      IF .NOT. EMPTY(XArray)
         RETURN XArray[i]
      ELSE
         RETURN .F.
      ENDIF
   RETURN
ENDCLASS
```

Parameters doorgeven aan hoofdklassen

In Hoofdstuk 11 wordt beschreven hoe u parameters definieert voor een klasse. Als parameters worden gedefinieerd door klassen in een hiërarchie, kunt u deze parameters doorgeven van subklassen naar hoofdklassen. Naast de eigen parameters (indien aanwezig), moet elke subklasse de parameters ontvangen die worden vereist door de bijbehorende hoofdklasse. Hier volgt een eenvoudig voorbeeld:

```
CLASS Kwadraat(n)                          && Parameter n definiëren
   this.Getal = n
   this.Kwadraat = n*n
ENDCLASS

CLASS Kubus(n) OF Kwadraat(n)              && n definiëren in de subklasse
   this.KubusGetal = n*this.KwadraatGetal
ENDCLASS

X = NEW Kwadraat(10)                       && Nieuw object Vierkant maken, 10 doorgeven
? X.KwadraatGetal                          && Resulteert in 100
Y = NEW Kubus(20)                          && Nieuw object Kubus maken, 20 doorgeven
? Y.KubusGetal                             && Resulteert in 8000
```

Aan de hand van de voorbeelden in de volgende sectie ziet u hoe parameters worden doorgegeven tussen klassen in een hiërarchie van klassen.

Een hiërarchie van klassen samenstellen

Afgeleide subklassen bieden het meeste vordeel als u een hiërarchie van klassen samenstelt waarbij de functionaliteit van elke subklasse voor het grootste deel wordt overgenomen van een hoofdklasse. In de subklasse kan vervolgens de functionaliteit worden gewijzigd door overgenomen leden te vervangen of nieuwe leden toe te voegen. Het gevolg is dat de code gemakkelijk is bij te werken en in hoge mate opnieuw bruikbaar is.

In Hoofdstuk 12 wordt beschreven hoe u programmataken abstract definieert om klassehiërarchieën te ontwikkelen. Taken groeperen in klassen en de klassen indelen in hiërarchieën zijn kerntaken bij object-georiënteerd ontwerpen. Hierna volgt een voorbeeld van de manier waarop u een eenvoudige klassehiërarchie implementeert.

U ontwerpt bijvoorbeeld een applicatie met meerdere formulieren. Elk formulier bevat knoppen. U wilt dat uw gebruikersinterface een consistent uiterlijk heeft en u ontwerpt

daarom een stijl voor de knoppen. In de volgende klassedefinitie worden de kenmerken van alle knoppen gedefinieerd.

```
CLASS MijnKnop(Form, Top, Left) OF PUSHBUTTON(Form)
   this.Top = Top              && Toegewezen van parameter Top
   this.Left = Left            && Toegewezen van parameter Left
   this.text = "Knop"          && Standaardtekst die moet worden vervangen
   this.FontName = "Arial"     && Font voor alle knoppen
   this.FontBold = .T.         && Alle knoppen zijn vet
   this.FontWidth = 6          && Fontbreedte voor alle knoppen

*-- eenvoudige methode definiëren als codeblok om de breedte van een knop
*-- in te stellen nadat u de feitelijke tekst hebt toegewezen
   this.SetWidth = {;this.width = LEN(this.text)+2}
ENDCLASS
```

Klasse MijnKnop dient als hoofdklasse voor alle knoppen. Voor sommige knoppen wilt u dat een geluidssignaal wordt gegeven als op de knop wordt geklikt. Met de volgende subklasse, afgeleid van MijnKnop, definieert u het geluidssignaal als standaardkenmerk dat u aan knoppen kunt toevoegen.

```
CLASS BiepKnop(Form, Top, Left) OF MijnKnop(Form, Top, Left)
   this.text = "Signaal"       && Standaardtekst voor knoppen met signaal
   this.Breedte()              && Overgenomen methode aanroepen om breedte in te stellen
   PROCEDURE OnClick              && Een procedure toewijzen die signaal geeft
      ? CHR(7)
   RETURN
ENDCLASS
```

Stel nu dat u een knop wilt maken die niet alleen een signaal genereert als u erop klikt, maar die bovendien wordt verplaatst. In de volgende subklasse, afgeleid van BiepKnop, wordt de functionaliteit van BiepKnop overgenomen en uitgebreid.

```
CLASS BiepVerplaats(Form, Top, Left) OF BiepKnop(Form, Top, Left)
   this.text = "Signaal en Verplaatsen"    && Standaardtekst voor bewegende
                                           && signaalknoppen
   this.Breedte()                          && Overgenomen methode aanroepen om
                                           && breedte in te stellen

   PROCEDURE OnClick                       && Gewijzigde versie van overgenomen
                                           && OnClick definiëren
      BiepKnop::OnClick()                  && OnClick aanroepen vanuit BiepKnop
      this.left = this.left+1              && Nieuwe functionaliteit toevoegen:
                                           && met 1 verplaatsen
   RETURN
ENDCLASS
```

In de volgende subklasse wordt een signaalknop gedefinieerd die groter wordt wanneer erop wordt geklikt. Het idee is hetzelfde als bij BiepVerplaats, maar de wijziging wordt anders afgehandeld. Eerst wordt OnClick vervangen door een codeblok in plaats van een procedure. Vervolgens wordt de OnClick met het signaal met *super* aangeroepen in plaats van met de feitelijke klassenaam.

```
CLASS BiepGroter(Form, Top, Left) OF BiepKnop(Form, Top, Left)
   this.text = "Signaal en Vergroten"
   this.Breedte()
   this.OnClick =
{;Super::OnClick();this.Width = This.Width + 1}
ENDCLASS
```

Hoofdstuk

12

Object-georiënteerd ontwerpen

Object-georiënteerd ontwerpen is een van de basismethoden om applicaties te ontwikkelen. In dBASE voor Windows wordt object-georiënteerd ontwerpen ondersteund doordat u objecten kunt maken, klassen definiëren en subklassen kunt definiëren op basis van hoofdklassen. In dit hoofdstuk komen de volgende onderwerpen aan de orde:

- Projecten die het meest geschikt zijn voor object-georiënteerd ontwerpen
- Inleiding tot enkele van de concepten die ten grondslag liggen aan de syntaxis en technieken die worden beschreven in Hoofdstuk 9, 10 en 11, zodat u deze technieken efficiënter kunt gebruiken bij het ontwikkelen van applicaties
- Toepassing van concepten voor object-georiënteerd ontwerpen op het ontwerp van een applicatie

In dit hoofdstuk wordt ervan uitgegaan dat u de vorige drie hoofdstukken, Hoofdstuk 9 tot en met 11, hebt doorgenomen. Zie het desbetreffende hoofdstuk als u meer wilt weten over termen die in dit hoofdstuk worden gebruikt.

Wanneer gebruikt u object-georiënteerd ontwerpen?

Object-georiënteerd ontwerpen is meestal de beste strategie voor het ontwerpen van applicaties, hoewel u deze strategie niet voor alle projecten hoeft te gebruiken. Als u eenvoudige invoerformulieren maakt, kunt u objecten maken en de bijbehorende kenmerken instellen zonder dat u hoeft na te denken over een "object-georiënteerde" benadering. In Formulierontwerp wordt dat voor u gedaan doordat een klassedefinitie voor het formulier wordt gegenereerd.

Hèt probleem dat u kunt oplossen met object-georiënteerd ontwerpen is *complexiteit*. Hoe complexer een programmeerproject wordt, des te groter de noodzaak om gebruik te maken van object-georiënteerd ontwerpen.

Object-georiënteerd ontwerpen is bij uitstek geschikt in de volgende situaties:

- *Grotere projecten.* Met object-georiënteerde technieken kunt u gemakkelijker een model maken van de realiteit. Complexe problemen kunnen in uw code eenvoudiger worden voorgesteld en kunnen gemakkelijker worden opgelost door de programmeur.
- *Projecten met meerdere programmeurs.* Omdat subroutines (methoden) en variabelen (kenmerken) "verborgen" zijn in objecten, is de kans kleiner dat het werk van de ene programmeur dat van de andere verstoort. Hierdoor wordt het voor programmeurs eenvoudiger om gemeenschappelijk gebruik te maken van codes.
- *Projecten die vaak worden gewijzigd.* Object-georiënteerde code is gemakkelijker te onderhouden en bij te werken. Zodra een klasse is geschreven en de fouten erin zijn opgespoord en verholpen, kunt u de functionaliteit van de klasse eenvoudig wijzigen door een subklasse af te leiden zonder dat u daarvoor de oorspronkelijke code hoeft aan te passen.

Overstappen van procedureel naar object-georiënteerd ontwerpen

Misschien bent u van mening dat het gemakkelijker is voor een onervaren programmeur om object-georiënteerd ontwerpen te leren dan voor een ervaren programmeur die gewend is aan procedureel ontwerpen. Dat is maar ten dele waar. In dBASE voor Windows is object-georiënteerd ontwerpen een natuurlijke uitbreiding van procedurele principes.

Bedenk dat object-georiënteerd ontwerpen een methode is om applicaties te *plannen en organiseren*, niet om deze te coderen. Als u bekend bent met de dBASE-taal, zijn er maar een paar nieuwe commando's en sleutelwoorden die u moet leren om te kunnen werken met objecten en klassen. De meeste subroutines die u in dBASE voor Windows schrijft, hebben veel weg van de subroutines die u hebt geschreven in vorige versies van dBASE. Het verschil schuilt in de wijze waarop de subroutines geheugenvariabelen gebruiken en opnieuw te gebruiken zijn.

De procedurele grondslagen

dBASE-programmeurs hebben verschillende achtergronden en hun opvattingen over programmeren lopen sterk uiteen. Alle dBASE-programmeurs volgen uiteindelijk echter een gemeenschappelijk ontwerpprincipe, namelijk dat van *modulariteit*. Dat wil zeggen, u kunt een programma van 300 regels schrijven dat sequentieel vanaf het begin tot het einde wordt uitgevoerd. Het is echter eenvoudiger om de taken van het programma te verdelen in modulen. Het programma van 300 regels wordt dan een hanteerbaar programma van 30 regels waarin meerdere subroutines worden aangeroepen.

Bij de verdeling van taken in subroutines kiezen de meeste programmeurs een benadering in twee fasen:

1 Begin met de meeste algemene taak en verdeel deze in steeds kleinere componenten. Hierdoor ontleedt u het probleem *van boven naar beneden.*

2 Breng de componenten onder in categorieën en groepeer de componenten in een hiërarchische structuur. Hierdoor voegt u de componenten *van beneden naar boven* samen.

Programmeurs die gebruik maken van object-georiënteerd ontwerpen, voeren dezelfde stappen uit maar met een ander doel. Een procedurele programmeur verdeelt het probleem in procedures en voegt de procedures samen tot een applicatie. Een object-georiënteerde programmeur verdeelt het probleem in klassen en voegt hiërarchieën van klassen samen tot een applicatie.

In de volgende twee secties worden enkele redenen voor deze verschillende benaderingen uitgelegd.

Problemen met het bereik van geheugenvariabelen

De subroutines in een procedurele applicatie manipuleren gegevens, waarbij sommige waarden tijdelijk worden opgeslagen in geheugenvariabelen.

- Variabelen van het type PUBLIC zijn beschikbaar in elke subroutine. In een applicatie met veel subroutines is derhalve voortdurend het gevaar aanwezig dat een reeds gedefinieerde variabele wordt overschreven door een variabele met dezelfde naam.
- Variabelen van het type PRIVATE hebben een beperkter bereik (deze variabelen zijn alleen beschikbaar in de subroutine waarin de variabelen worden gemaakt, en in elke subroutine die daarna wordt aangeroepen). Ook bij deze variabelen bestaat het gevaar dat de variabelen worden overschreven in applicaties met meerdere niveaus van geneste programma's.

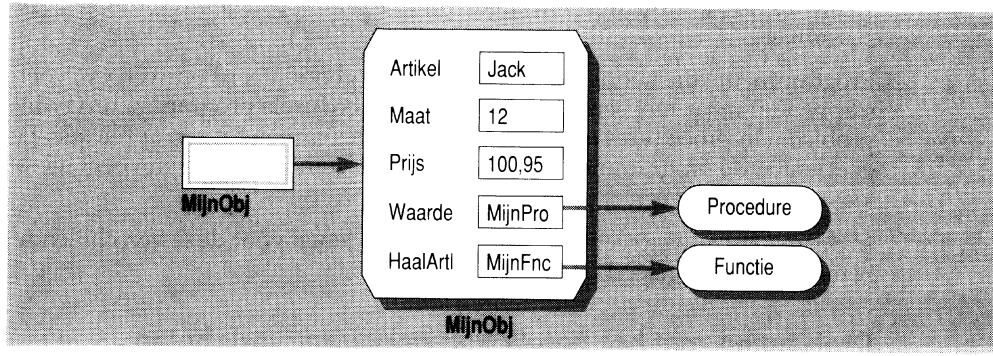
De kans op conflicten tussen geheugenvariabelen wordt nog groter als meerdere programmeurs aan hetzelfde project werken en de namen van variabelen moeten worden gecoördineerd. Als u procedurebibliotheken van derden gebruikt, is het mogelijk dat variabelen in die procedures strijdig zijn met uw eigen variabelen.

dBASE voor Windows kent twee nieuwe manieren om een bereik op te geven: LOCAL en STATIC. Hiermee kunnen conflicten tussen geheugenvariabelen worden voorkomen. Variabelen die worden gedefinieerd als LOCAL of STATIC, zijn alleen beschikbaar in de subroutine waarin ze worden gedefinieerd. Elke subroutine in een applicatie kan derhalve een variabele X gebruiken zonder dat er conflicten ontstaan, zolang X telkens wordt gedefinieerd als STATIC of LOCAL. (Zie Hoofdstuk 5 voor meer informatie over geheugenvariabelen van het type STATIC of LOCAL.)

Geheugenvariabelenworden echter op een betere manier beschermd als deze worden ingekapseld in subroutines door objecten te maken. In dat geval verwijzen de subroutines niet naar geheugenvariabelen, maar naar kenmerken van objecten.

Inkapselen houdt in dat gegevens worden gecombineerd met de subroutines die de gegevens gebruiken, zodat een nieuwe structuur wordt gevormd: een object. Door inkapseling worden gegevens in objecten verborgen, zodat de gegevens worden beschermd tegen gegevens met dezelfde naam op een andere plaats in de applicatie. In Afbeelding 12.1 ziet u hoe geheugenvariabelen en procedures in objecten worden ingekapseld.

Afbeelding 12.1 Code en gegevens inkapselen in object-georiënteerde applicaties



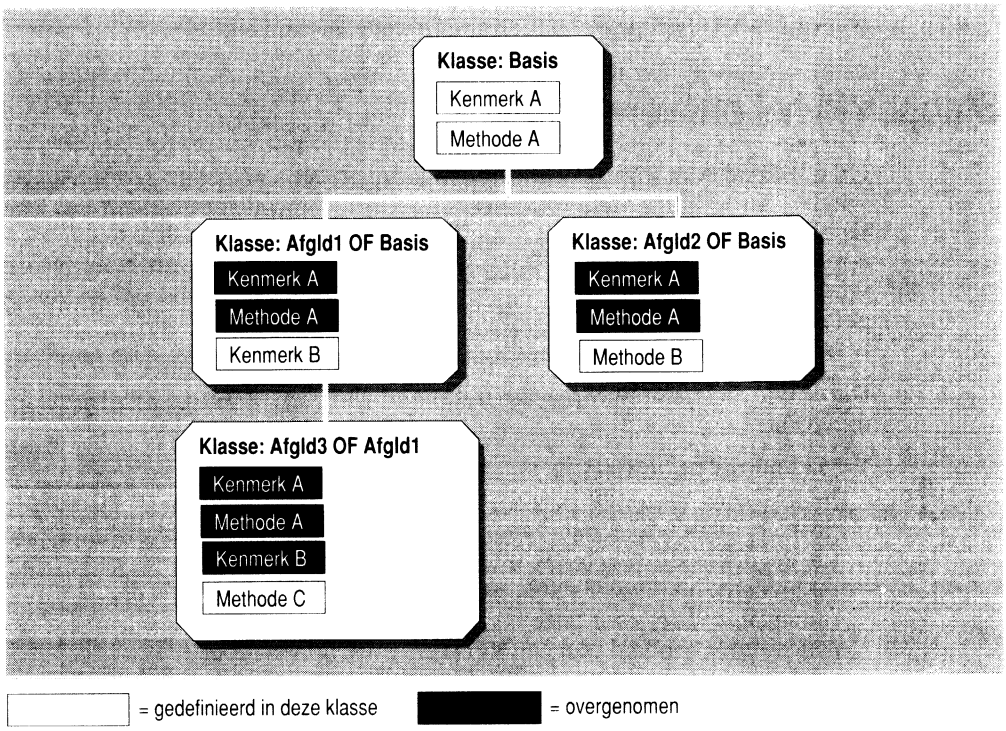
Problemen bij opnieuw bruikbaar maken van subroutines

Programmeurs streven ernaar subroutines te schrijven die zij in meerdere situaties kunnen gebruiken, zoals een procedure voor meldingen waarmee fouten, waarschuwingen of Help-informatie kan worden weergegeven. In veel gevallen is een procedure echter niet zonder meer opnieuw bruikbaar, maar moet deze eerst worden gewijzigd. Als een programmeur wijzigingen aanbrengt in de procedure, ontstaat het risico dat nieuwe fouten worden gemaakt waardoor de procedure niet meer werkt.

Als u code schrijft die opnieuw moet worden gebruikt, kunt u beter klassen definiëren waarbij elke klasse de definities van kenmerken en procedures (methoden) bevat die worden gebruikt door objecten, die op basis van de klasse zijn gemaakt.

Overname houdt in dat een klasse wordt gedefinieerd en vervolgens gebruikt om een hiërarchie van afgeleide objecten te maken, waarbij elk afgeleid object de gegevens en subroutines van het hoofdobject "overneemt". In Afbeelding 12.2 ziet u een voorbeeld van overname in een klassehiërarchie.

Afbeelding 12.2 Een voorbeeld van klassehiërarchie



Overname komt er in grote lijnen op neer dat u programmeert door gedrag te wijzigen. Verschillende objecten die gegevens of gedrag gemeenschappelijk hebben, kunnen die informatie overnemen van een bestaand object. U hoeft dan alleen de benodigde wijzigingen aan te brengen. Als u gebruik maakt van overname, is uw code gemakkelijker opnieuw te gebruiken.

Als u een klasse enigszins moet aanpassen voor een ander gebruik, wijzigt u niet de klassedefinitie zelf. In plaats daarvan leidt u een subklasse van de klasse af en vervangt u alleen de kenmerken of methoden die u moet wijzigen. Objecten die zijn gemaakt op basis van de subklasse, kunnen op dezelfde manier worden gebruikt als objecten die zijn gemaakt op basis van de hoofdklasse. Alleen het gedrag van de objecten is anders.

Polymorfisme betekent dat een object in een hiërarchie van klassen de mogelijkheid wordt gegeven om een handeling te implementeren die alle objecten in de hiërarchie gemeen hebben, maar op de meest geschikte wijze voor het object. Polymorfisme (letterlijk: "met vele gedaanten") betekent dat objecten in een hiërarchie dezelfde subroutine op naam kunnen delen, maar dat de subroutine in elk object een andere "gedaante" kan aannemen.

Aan de hand van het dBASE-commando SKIP wordt duidelijk wat de voordelen zijn van polymorfisme. De werking van SKIP is afhankelijk van de status van het huidige werkgebied. Als er geen index actief is, gaat SKIP naar het volgende hoogste recordnummer. Als er een index voor het werkgebied is, gaat SKIP naar het volgende

record in de index. Als er een filter is ingesteld, gaat SKIP naar het volgende record dat aan de filtervoorwaarde voldoet. U hoeft SKIP geen instructies te geven omtrent het werkgebied. U typt gewoon `SKIP`, en de rest gaat vanzelf.

Stappen in het ontwerpproces

Nu u bekend bent met de achterliggende concepten van object-georiënteerd ontwerpen, wilt u waarschijnlijk weten hoe het ontwerpen van een applicatie in zijn werk gaat. Ongeacht de gebruikte ontwerpmethode worden in het ontwikkelingsproces van een applicatie altijd de volgende stappen gevolgd: de situatie wordt geanalyseerd en verdeeld in componenten, de componenten worden georganiseerd en vervolgens geïmplementeerd. In object-georiënteerd ontwerpen zijn klassen de componenten waarmee u de applicatie samenstelt.

Dit zijn de algemene stappen die u uitvoert als u een object-georiënteerde applicatie ontwerpt:

1 **De klassen identificeren.**
In deze stap, waarmee procedurele programmeurs vertrouwd zijn, wordt een probleem van boven naar beneden ontleed. Analyseer de vereisten voor uw project (de gegevens die moeten worden bijgehouden, de taken die moeten worden uitgevoerd, de uitvoer die moet worden gegenereerd) en stel een lijst met klassen samen als model voor deze vereisten. In deze betekenis is een klasse elke fysieke of conceptuele eenheid met een duidelijk omschreven doel.

   De volgende elementen komen voor deze klasse in aanmerking:

   - Groepen gegevens in tabellen, zoals klanten, bestellingen of inventaris.
   - Interfaces voor gegevens (meestal formulieren) voor taken als het toevoegen, bewerken of weergeven van gegevens. Een groot deel van de klassen die u identificeert, is gebaseerd op een formulier.
   - Speciale taken, zoals samenvatten, compileren of zoeken naar gegevens.

2 **De gegevens en met de klassen gerelateerde taken identificeren.**
Som voor elke klasse de gegevens op die door de klasse worden gebruikt, alsmede de taken die de klasse met deze informatie uitvoert. Welke gegevens worden door de formulieren weergegeven? Moet de gebruiker elementen toevoegen of bewerken?

3 **De relaties tussen de klassen identificeren.**
Probeer gemeenschappelijke elementen tussen de klassen te vinden. Sommige klassen staan mogelijk op zichzelf, terwijl andere zijn georganiseerd in een hiërarchie. Als bijvoorbeeld drie klassen dezelfde of soortgelijke taken uitvoeren, groepeert u deze klassen.

   Sommige klassen in de hiërarchie dienen mogelijk alleen als hoofdklassen. Uit deze klassen worden waarschijnlijk nooit objecten gemaakt. Als twee klassen bijvoorbeeld dezelfde gegevens bevatten, met uitzondering van een paar elementen, kunt u overwegen een derde klasse te maken met de gegevens die de twee andere klassen gemeenschappelijk hebben. Deze klassen worden dan subklassen van de derde klasse en nemen de gegevens over die deze gemeenschappelijk hebben.

U streeft ernaar de taken en gegevens die de klassen gemeen hebben, te "abstraheren" om dubbele elementen te voorkomen. Het resultaat is een hiërarchie van klassen.

4 **De klassen implementeren.**
Omdat in een groot deel van uw klassen formulieren worden gemaakt, kunt u de klassedefinities genereren met Formulierontwerp. Schrijf een klassedefinitie met behulp van CLASS...ENDCLASS voor andere klassen die niet zijn gebaseerd op formulieren.

Deze stappen in het ontwerpproces zijn niet de enig mogelijke. U kunt deze stappen net zo nauwgezet volgen als u wilt. Bedenk echter dat u bij het ontwerpen van actie-gestuurde applicaties veel meer problemen het hoofd zult moeten bieden dan bij het ontwerpen van menu-gestuurde applicaties. De voldoening die u ervaart als u zich de technieken voor object-georiënteerd ontwerpen in de Windows-omgeving hebt eigen gemaakt, is groot. De gevaren die op de loer liggen als u genoegen neemt met een slecht ontwerp, zijn echter nog veel groter.

Een voorbeeld

Met het scenario in deze sectie doorloopt u stap voor stap een codevoorbeeld van een object-georiënteerde applicatie.

U schrijft programma's voor een klein postorderbedrijf. U moet informatie bijhouden over werknemers (zowel full-time als part-time) en elke week hun salaris berekenen. U moet ook een adressenlijst bijhouden van klanten die een bestelling plaatsen en van potentiële klanten die nog niets hebben besteld.

1 **De klassen identificeren.**
De belangrijkste klassen van informatie zijn:
- Full-time werknemers
- Part-time werknemers
- Klanten
- Potentiële klanten

2 **De gegevens en de taken identificeren die met elke klasse worden geassocieerd.**
Voor elke klasse hebt u een interface nodig voor elementaire bewerkingen en weergavedoeleinden. Voor de werknemers moet u elke week het salaris berekenen.

In Tabel 12.1 wordt een overzicht gegeven van de gegevens en taken die bij elke klasse behoren:

Tabel 12.1 Gegevens en taken voor voorbeeldklassen

Klasse	Gegevens	Taken
Full-time werknemers	Naam en adres Bonussen Salaris	Gegevens weergeven en bewerken. Weekloon berekenen.
Part-time werknemers	Naam en adres Uurloon	Gegevens weergeven en bewerken. Weekloon berekenen.
Klanten	Naam en adres Aantal bestellingen	Gegevens weergeven en bewerken.
Potentiële klanten	Naam en adres Type bedrijf	Gegevens weergeven en bewerken.

3 De relaties tussen klassen identificeren.

Omdat de vier klassen alle zijn gebaseerd op formulieren en omdat deze klassen informatie bevatten over naam en adres, definieert u een hoofdklasse BasisForm waarvan alle andere klassen worden afgeleid. BasisForm bevat invoervakken voor het weergeven van namen en adressen. Bovendien stelt BasisForm de formulierkenmerken in die alle formulieren gemeen hebben, zoals FontName. Dit is een voorbeeld van overname.

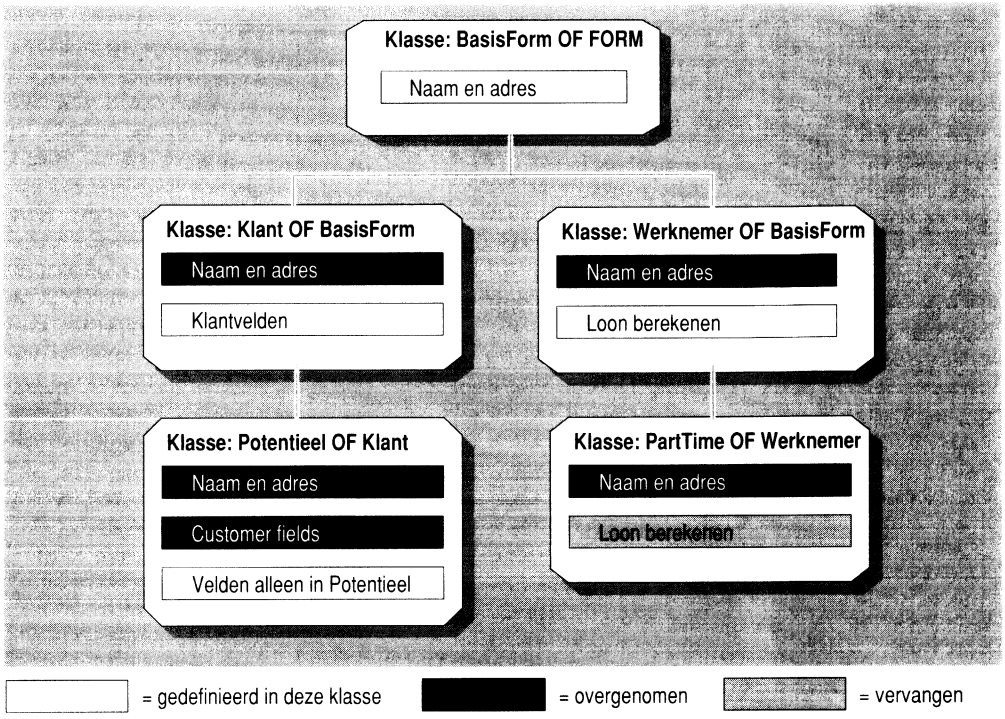
Vervolgens moet u voor potentiële klanten dezelfde informatie opslaan als voor vaste klanten en moet u tevens aanvullende informatie opslaan. U maakt niet een hele nieuwe klasse, maar een klasse Potentieel die is gebaseerd op de klasse Klant. De klasse Potentieel neemt de kenmerken van de klasse Klant over. Vervolgens voegt u invoervakken toe voor de velden die alleen worden weergegeven in de tabel Potentieel .

Vervolgens moet u dezelfde informatie opslaan voor full-time en part-time werknemers, met als verschil dat het salaris van part-timers anders wordt berekend. Definieer eerst een klasse Werknemer en vervolgens een klasse PartTime die is gebaseerd op de klasse Werknemer. De klasse Werknemer bevat een subroutine, BerekenLoon(), voor het berekenen van het jaarsalaris. In de klasse PartTime kunt u BerekenLoon() vervangen door een andere functie waarmee het salaris wordt berekend op basis van gewerkte uren en waarmee in het voorbeeld geen vakantiegeld wordt berekend.

Dit is een voorbeeld van polymorfisme. Uw programma kan het salaris van een werknemer op dezelfde manier ophalen, door BerekenLoon() aan te roepen ongeacht het type werknemer. Als u de implementatie van de taak verbergt, kunt u de details van de implementatie wijzigen zonder dat daarvoor wijzigingen in de interface nodig zijn. Stel dat de full-timers elk kwartaal een bonus krijgen. In dat geval wijzigt u BerekenLoon() voor de klasse Werknemer. Met polymorfisme is de code in hoge mate opnieuw bruikbaar.

In Afbeelding 12.3 wordt de hiërarchie van klassen weergegeven:

Afbeelding 12.3 Klassehiërarchie voor voorbeeldscenario



4 De klassen implementeren.
Hierna wordt de broncode weergegeven waarmee de klassen worden geïmplementeerd.

```
* POSTORDR.PRG
LOCAL F
F = NEW FULLTIME ()
F.OPEN()
P = NEW PARTTIME ()
P.OPEN()
C = NEW KLANT()
C.OPEN()
R = NEW POTENTIEEL()
R.OPEN()
```

```
***---------------------------- CLASS: BasisForm ------------------------------**
CLASS BASISFORM OF FORM
   this.Text = "Formulier"
   this.Width =          91.03
   this.Top =          0.05
   this.Left =          3.18
   this.Height =         24.83
   this.Minimize = .F.
   this.Maximize = .F.
   this.HelpFile = ""
   this.HelpId = ""
   DEFINE TEXT TF_NAAM OF THIS;
       PROPERTY;
         ColorNormal "N/W",;
         Text "Naam",;
         Width          7.95,;
         Top          3.28,;
         Left          11.93,;
         Height          1.09,;
         Border .F.
   DEFINE TEXT TF_ADRES OF THIS;
       PROPERTY;
         ColorNormal "N/W",;
         Text "Adres",;
         Width          7.95,;
         Top          5.47,;
         Left          11.93,;
         Height          1.18,;
         Border .F.
   DEFINE TEXT TF_PLAATS OF THIS;
       PROPERTY;
         ColorNormal "N/W",;
         Text "Plaats",;
         Width          7.95,;
         Top          9.68,;
         Left         20.86,;
         Height          1.09,;
         Border .F.
   DEFINE TEXT TF_REGIO OF THIS;
       PROPERTY;
         ColorNormal "N/W",;
         Text "Regio",;
         Width          7.95,;
         Top          9.68,;
         Left         54.06,;
         Height          1.09,;
         Border .F.
```

```
DEFINE TEXT TF_POSTCODE OF THIS;
    PROPERTY;
      ColorNormal "N/W",;
      Text "Postcode",;
      Width          10.53,;
      Top          9.59,;
      Left         65.38,;
      Height          1.18,;
      Border .F.
DEFINE ENTRYFIELD EF_NAAM OF THIS;
    PROPERTY;
      ColorNormal "",;
      Width         59.63,;
      Top          3.20,;
      Left         20.27,;
      Height          1.43,;
      Border .T.,;
      Value "",;
      ColorHighLight ""
DEFINE ENTRYFIELD EF_ADRES OF THIS;
    PROPERTY;
      ColorNormal "",;
      Width         59.23,;
      Top          5.30,;
      Left         20.27,;
      Height          1.43,;
      Border .T.,;
      Value "",;
      ColorHighLight ""
DEFINE ENTRYFIELD EF_PLAATS OF THIS;
    PROPERTY;
      ColorNormal "",;
      Width         28.62,;
      Top          7.83,;
      Left         20.27,;
      Height          1.34,;
      Border .T.,;
      Value "",;
      ColorHighLight ""
DEFINE ENTRYFIELD EF_REGIO OF THIS;
    PROPERTY;
      ColorNormal "",;
      Width          4.77,;
      Top          7.83,;
      Left         54.25,;
      Height          1.34,;
      Border .T.,;
      Value "",;
      ColorHighLight ""
```

```
   DEFINE ENTRYFIELD EF_POSTCODE OF THIS;
       PROPERTY;
         ColorNormal "",;
         Width           13.75,;
         Top           7.75,;
         Left           65.19,;
         Height           1.43,;
         Border .T.,;
         Value "",;
         ColorHighLight ""
ENDCLASS

***----------------------------- CLASS: Werknemer -----------------------------**
CLASS WERKNEMER OF BASISFORM
   DEFINE ENTRYFIELD EF_TARIEF OF THIS;
       PROPERTY;
         ColorNormal "",;
         Width           13.91,;
         Top          15.07,;
         Left          20.07,;
         Height           1.43,;
         Border .T.,;
         Value "",;
         ColorHighLight ""
ENDCLASS

***----------------------------- CLASS: PartTime ------------------------------**
CLASS PARTTIME OF WERKNEMER
   this.text = "Informatie over part-time werknemers"
   DEFINE TEXT TF_SALARIS OF THIS;
       PROPERTY;
         ColorNormal "N/W",;
         Text "Uurloon",;
         Width          9.30,;
         Top          15.07,;
         Left           9.59,;
         Height           1.26,;
         Border .F.
   DEFINE TEXT TF_TWEE_WKL OF THIS;
       PROPERTY;
         ColorNormal "N/W",;
         Text "Brutoloon per betalingsperiode",;
         Width          30.02,;
         Top          14.90,;
         Left          52.28,;
         Height           1.52,;
         Border .F.
```

```
  DEFINE TEXT TF_BETALING OF THIS;
      PROPERTY;
        ColorNormal "N/W*",;
        Text "",;
        Width          11.53,;
        Top          17.17,;
        Left          61.61,;
        Height           1.18,;
        Border .T.
  DEFINE TEXT TF_UREN OF THIS;
      PROPERTY;
        ColorNormal "N/W",;
        Text "Uren",;
        Width           6.95,;
        Top          18.10,;
        Left           12.00,;
        Height           1.09,;
        Border .F.
  DEFINE ENTRYFIELD EF_UREN OF THIS;
      PROPERTY;
        ColorNormal "",;
        Width          13.91,;
        Top          17.85,;
        Left          20.48,;
        Height           1.43,;
        Border .T.,;
        Value "",;
        ColorHighLight ""
  DEFINE PUSHBUTTON PB_BRKNLOON OF THIS;
      PROPERTY;
        OnClick CLASS::PB_BRKNLOON_ONCLICK,;
        ColorNormal "N/W",;
        Text "Loon berekenen",;
        Width          25.63,;
        Top          21.29,;
        Left          62.41,;
        Height           3.20,;
        Group .T.
Procedure PB_BRKNLOON_OnClick
TEXTPAY = STR(VAL(FORM.EF_TARIEF.VALUE) * VAL(FORM.EF_UREN.VALUE))
FORM.TF_BETALING.TEXT = TEXTPAY
ENDCLASS
```

```
***------------------------------ CLASS: FullTime ------------------------------**
CLASS FULLTIME OF WERKNEMER
   this.text = "Informatie over full-time werknemers"
   this.left = 95.40
   DEFINE TEXT TF_SALARIS OF THIS;
       PROPERTY;
         ColorNormal "N/W",;
         Text "Jaarsalaris",;
         Width         11.30,;
         Top         15.07,;
         Left          7.59,;
         Height          1.26,;
         Border .F.
   DEFINE TEXT TF_BONUS OF THIS;
       PROPERTY;
         ColorNormal "N/W",;
         Text "Bonussen",;
         Width          10.53,;
         Top         19.02,;
         Left         4.17,;
         Height          1.09,;
         Border .F.
   DEFINE CHECKBOX CB_ZIEKTE OF THIS;
       PROPERTY;
         ColorNormal "N/W",;
         Text "Ziektekostenverzekering",;
         Width         26.84,;
         Top         19.27,;
         Left         19.88,;
         Height          1.60,;
         Value .F.,;
         Group .T.
   DEFINE CHECKBOX CB_TANDARTS OF THIS;
       PROPERTY;
         ColorNormal "N/W",;
         Text "Tandartsverzekering",;
         Width         25.44,;
         Top         21.29,;
         Left         19.88,;
         Height          1.60,;
         Value .F.,;
         Group .T.
   DEFINE CHECKBOX CB_401K OF THIS;
       PROPERTY;
         ColorNormal "N/W",;
         Text "401K-plan",;
         Width         17.68,;
         Top         23.15,;
         Left         19.88,;
         Height          1.60,;
         Value .T.,;
         Group .T.
```

```
   DEFINE TEXT TF_TW_WKL OF THIS;
       PROPERTY;
         ColorNormal "N/W",;
         Text "Brutoloon per betalingsperiode",;
         Width         30.02,;
         Top          14.98,;
         Left          52.28,;
         Height           1.52,;
         Border .F.
   DEFINE TEXT TF_BETALING OF THIS;
       PROPERTY;
         ColorNormal "N/W*",;
         Text "",;
         Width           11.53,;
         Top          17.17,;
         Left          61.61,;
         Height           1.18,;
         Border .T.
   DEFINE PUSHBUTTON PB_BRKNLOON OF THIS;
       PROPERTY;
         OnClick CLASS::PB_BRKNLOON_ONCLICK,;
         ColorNormal "N/W",;
         Text "Loon berekenen",;
         Width          25.63,;
         Top          21.29,;
         Left          62.41,;
         Height           3.20,;
         Group .T.

   PROCEDURE PB_BRKNLOON_OnClick
      form.TF_BETALING.TEXT = STR(VAL(form.EF_TARIEF.VALUE)/26)
   RETURN
ENDCLASS

***------------------------------ CLASS: Klant ------------------------------**
CLASS KLANT OF BASISFORM
   this.text = "Informatie over klanten"
   this.top = 27.27
   DEFINE LINE LIJN1 OF THIS;
       PROPERTY;
         ColorNormal "N",;
         Width           1.59,;
         Top          12.96,;
         Left           1.99,;
         Bottom          12.96,;
         Right          56.00
```

```
   DEFINE TEXT TF_AANTORDERS OF THIS;
       PROPERTY;
         ColorNormal "N/W",;
         Text "Aantal orders",;
         Width         20.07,;
         Top         14.90,;
         Left          5.37,;
         Height          1.43,;
         Border .F.
   DEFINE ENTRYFIELD EF_AANTORDERS OF THIS;
       PROPERTY;
         ColorNormal "",;
         Width          10.73,;
         Top         14.82,;
         Left         26.63,;
         Height          1.43,;
         Border .T.,;
         Value "",;
         ColorHighLight ""
ENDCLASS

***------------------------------ CLASS: Potentieel ------------------------------**
CLASS POTENTIEEL OF KLANT
   this.text = "Informatie over potentiële klanten"
   this.left = 95.40
   DEFINE TEXT TF_VERWACHT OF THIS;
       PROPERTY;
         ColorNormal "N/W",;
         Text "verwacht",;
         Width          12.53,;
         Top         16.16,;
         Left          5.37,;
         Height          1.09,;
         Border .F.
   DEFINE TEXT TF_BEDRTYPE OF THIS;
       PROPERTY;
         ColorNormal "N/W",;
         Text "Type bedrijf",;
         Width         19.48,;
         Top         19.19,;
         Left          5.17,;
         Height          1.18,;
         Border .F.
```

```
    DEFINE ENTRYFIELD EF_BEDRTYPE OF THIS;
        PROPERTY;
          ColorNormal "",;
          Width         39.15,;
          Top          19.19,;
          Left          26.63,;
          Height           1.43,;
          Border .T.,;
          Value "",;
          ColorHighLight ""
ENDCLASS
```

Hoe kom ik meer te weten?

In dBASE voor Windows kunt u voor het eerst met de dBASE-taal object-georiënteerd programmeren. *Aan de slag* bevat een bibliografie van publicaties over dBASE voor Windows. In een aantal van deze publicaties komt ook object-georiënteerd programmeren aan de orde.

Aan de hand van de volgende literatuur kunt u meer te weten komen over de principes van object-georiënteerd programmeren:

- *Object-Oriented Technology: A Manager's Guide,* door David A.Taylor. In dit boek krijgt u een aangenaam geschreven, conceptueel overzicht van object-georiënteerd ontwerpen en programmeerkwesties dat niet gebonden is aan een bepaalde databasetaal.
- *Designing Object-Oriented Software,* door Rebecca Wirfs-Brock, Brian Wilkerson en Lauren Wiener. De grondbeginselen van object-georiënteerd ontwerpen worden hierin stap voor stap behandeld.

Hoewel object-georiënteerde technieken nieuw zijn in dBASE, zijn deze technieken al langer gemeengoed in andere object-georiënteerde talen, zoals Borland C++ en Borland Pascal. Bestudering van de object-georiënteerde aspecten van deze talen kan u behulpzaam zijn bij het begrijpen van de implementatie in dBASE. U zult ontdekken dat sommige elementen in de dBASE-taal, zoals NEW(), *this* en ::, hun directe evenknie hebben in andere object-georiënteerde talen.

- *C++ Primer,* door Stanley B. Lippman, bevat heldere uitleg over de klassemechanismen die zijn geïmplementeerd in C++.

Formulieren

In dit deel worden verschillende invalshoeken belicht bij het samenstellen van applicaties met behulp van formulieren. Tevens komen programmeertechnieken voor het maken van formulieren aan bod. Raadpleeg het *Handboek* voor meer informatie over het gebruik van Formulierontwerp.

Wilt u alleen actie-afhandelingsroutines voor een formulier schrijven, dan raadpleegt u de volgende hoofdstukken:

1 Hoofdstuk 2, als inleiding voor actiegestuurde programma's

2 Hoofdstuk 14, "Acties afhandelen", met aanwijzingen en voorbeelden

Wilt u leren hoe u applicaties met behulp van formulieren samenstelt, dan raadpleegt u de volgende hoofdstukken:

1 Hoofdstuk 13, "Applicaties samenstellen met formulieren"

2 Hoofstuk 14 en 15, over het afhandelen van acties en het gebruik van eigen stuurelementen

Lees Hoofdstuk 16, "Werken met gegenereerde code", over het gebruik van tweeweg-hulpmiddelen, Formulierontwerp en Menu-ontwerp.

Dit deel bevat de volgende hoofdstukken:

- Hoofdstuk 13, "Applicaties samenstellen met formulieren"
- Hoofdstuk 14, "Acties afhandelen"
- Hoofdstuk 15, "Eigen stuurelementen maken"
- Hoofdstuk 16, "Werken met gegenereerde code"

liever met een eigen dialoogvenster. In Afbeelding 13.1 ziet u het invoerformulier (KlBewerken) en het opzoekvenster (KlZoeken).

Afbeelding 13.1 Voorbeelden van actie-afhandelingsroutines voor een invoerformulier en een dialoogvenster

Het formulier KlBewerken in Afbeelding 13.1 is een niet-modaal formulier. Gebruikers kunnen informatie invoeren in dit formulier of een ander formulier selecteren. Wanneer gebruikers drukken op **Zoeken**, moeten ze opgeven welke naam ze willen zoeken voordat de applicatie de zoekactie kan uitvoeren. Het formulier KlZoeken is dus een modaal formulier. Gebruikers moeten dit formulier sluiten voordat ze een andere handeling kunnen uitvoeren. Modale formulieren zijn dialoogvensters die informatie moeten ophalen voordat een niet-modaal formulier kan worden geactiveerd.

Een aanverwant kenmerk van formulieren is het kenmerk **MDI**. Een MDI-formulier heeft specifieke eigenschappen die voldoen aan de Windows-standaard Multiple Document Interface (MDI). Hiermee kunt u meerdere vensters openen binnen het toepassingsvenster. De meeste Windows-toepassingen gebruiken MDI om meerdere documenten of meerdere weergaven van hetzelfde document te beheren binnen het hoofdvenster van de toepassing. Als u uw formulier wilt weergeven in een MDI-venster, stelt u **MDI** in op .T. (waar).

MDI-vensters hebben onder meer de volgende eigenschappen:

- De vensters kunnen worden verplaatst en het formaat van de vensters kan worden gewijzigd, net als bij toepassingsvensters. Als het kenmerk **MDI** waar is, worden de kenmerken **Moveable** en **Sizeable** genegeerd.
- De vensters zijn subvensters van het toepassingvensters en worden weergegeven in het menu **Venster**, zelfs wanneer de vensters actief zijn.
- De vensters hebben het symbool **Systeemmenu**, de knoppen **Maximumvenster** en **Pictogram**, en een titelbalk met de naam van het venster. Als **MDI** waar is, worden de kenmerken **Minimize**, **Maximize** en **SysMenu** genegeerd.
- Wanneer de vensters actief zijn, vervangen de bijbehorende menu's de menu's op de menubalk van de applicatie. (Als **MDI** onwaar (.F.) is, verschijnen de menu's op de menubalk van het formulier.)

- De sneltoets om het formulier te sluiten is *Ctrl+F4* in plaats van *Alt+F4*. Met *Alt+F4* sluit u het toepassingsvenster en met *Ctrl+F4* sluit u het formulier.

Belangrijk Een MDI-formulier kan niet modaal zijn. Voordat u een modaal formulier opent, moet u het formulierkenmerk **MDI** instellen op .F. (onwaar).

Modale en niet-modale formulieren openen

U maakt modale en niet-modale formulieren op dezelfde manier, met uitzondering van het kenmerk **MDI**, dat .F. (onwaar) moet zijn voor modale formulieren.

- Gebruik de formuliermethode Open() of het commando OPEN FORM om een niet-modaal formulier weer te geven.
- Gebruik de formuliermethode ReadModal() of de functie READMODAL() om een modaal formulier weer te geven.

In Afbeelding 13.2 ziet u een codevoorbeeld voor een modaal en een niet-modaal formulier.

Afbeelding 13.2 Voorbeeldcode om modale en niet-modale formulieren te maken

Een modaal formulier

```
DEFINE FORM MijnMod FROM 10,10 TO 15,30 PROPERTY;
   MDI .F.
DEFINE TEXT Hallo OF MijnMod PROPERTY;
   Top 1,;
   Left 7,;
   Text "Hallo!"
DEFINE PUSHBUTTON TotZiens OF MijnMod PROPERTY;
   Top 3,;
   Left 5,;
   Text "Tot_ziens",;
   OnClick {;form.CLOSE()}

MijnMod.ReadModal()

? CHR(7)+CHR(7)+CHR(7)    && drie pieptonen
```

Een niet-modaal formulier

```
DEFINE FORM MijnNietMod FROM 10,10 TO 15,30

DEFINE TEXT Hallo OF MijnNietMod PROPERTY;
   Top 1,;
   Left 7,;
   Text "Hallo!"
DEFINE PUSHBUTTON TotZiens OF MijnNietMod PROPERTY;
   Top 3,;
   Left 5,;
   Text "Tot_ziens",;
   OnClick {;form.CLOSE()}

MijnNietMod.Open()

? CHR(7)+CHR(7)+CHR(7)    && drie pieptonen
```

Let op de volgende punten in dit voorbeeld:

- Het formulier MijnMod wordt geopend met de methode ReadModal(), en het formulier MijnNietMod wordt geopend met de methode Open(). U kunt MijnMod ook openen met de functie READMODAL() en MijnNietMod met het commando OPEN FORM.
- Het kenmerk **MDI** van MijnMod is .F. (onwaar), zodat het formulier geen MDI-venster is. MDI-vensters kunnen niet modaal zijn.

Modaliteit en programma-uitvoering

De uitvoering van een programma hangt niet alleen af van de modaliteit van het formulier, maar ook van de manier waarop u een formulier opent.

- Als u een modaal formulier opent, stopt de uitvoering van het programma totdat het formulier wordt gesloten
- Als u een niet-modaal formulier opent, wordt het formulier weergegeven en gaat de uitvoering van het programma door

In Afbeelding 13.2 staat de volgende regel op dezelfde plaats in beide codevoorbeelden, maar wordt de regel op verschillende momenten uitgevoerd:

```
? CHR(7)+CHR(7)+CHR(7)    && drie pieptonen
```

In het codevoorbeeld voor een modaal formulier stopt de uitvoering bij de regel `MijnMod.ReadModal()` en klinken de drie pieptonen nadat het formulier wordt gesloten. In het codevoorbeeld voor een niet-modaal formulier stopt de uitvoering niet en klinken de drie pieptonen onmiddellijk nadat het formulier is geopend.

Omdat de programma-uitvoering niet stopt, moet u zorgvuldig omspringen met de beschikbaarheid van geheugenvariabelen wanneer u niet-modale formulieren opent. Het programma waarmee een niet-modaal formulier wordt gemaakt, kan volledig worden uitgevoerd, zodat het formulier open en actief blijft. Het bereik van geheugenvariabelen die worden gebruikt door de methoden of actie-afhandelingsroutines van het formulier, moet zo worden ingesteld dat de variabelen beschikbaar zijn.

In plaats van traditionele geheugenvariabelen kunt u beter eigen kenmerken definiëren om tijdelijke waarden op te slaan. Een kenmerk is beschikbaar zolang het bijbehorende object bestaat, ongeacht het programma dat wordt uitgevoerd.

In Afbeelding 13.3 is de code voor het niet-modale formulier in Afbeelding 13.2 gewijzigd en wordt er een melding weergegeven wanneer u klikt op de knop. In de code aan de linkerkant wordt een PUBLIC-variabele gedefinieerd om de melding op te slaan. In de code aan de rechterkant wordt een eigen formulierkenmerk gemaakt (deze methode heeft de voorkeur).

Afbeelding 13.3 De beschikbaarheid van geheugenvariabelen in niet-modale formulieren

Een PUBLIC-variabele in een niet-modaal formulier

```
PUBLIC LaatsteWoord
LaatsteWoord = "Vaarwel wrede wereld!"

DEFINE FORM MijnNietMod FROM 10,10 TO 15,30 PROPERTY;
   MDI .T.

DEFINE TEXT Hallo OF MijnNietMod PROPERTY;
   Top 1,;
   Left 7,;
   Text "Hallo!"
DEFINE PUSHBUTTON TotZiens OF MijnNietMod PROPERTY;
   Top 3,;
   Left 5,;
   Text "Tot_ziens",;
   OnClick CloseProc,;
   Width 8

MijnNietMod.Open()

PROCEDURE CloseProc
   ? LaatsteWoord
   form.close()
RETURN
```

Een eigen kenmerk in een niet-modaal formulier

```
DEFINE FORM MijnNietMod FROM 10,10 TO 15,30 PROPERTY;
   MDI .T.;
   CUSTOM LaatsteWoord "Vaarwel wrede wereld!"

DEFINE TEXT Hallo OF MijnNietMod PROPERTY;
   Top 1,;
   Left 7,;
   Text "Hallo!"
DEFINE PUSHBUTTON TotZiens OF MijnNietMod PROPERTY;
   Top 3,;
   Left 5,;
   Text "Tot_ziens",;
   OnClick CloseProc,;
   Width 8

MijnNietMod.Open()

PROCEDURE CloseProc
   ? form.LaatsteWoord
   form.close()
RETURN
```

Het procedurebestand met procedures voor methoden of actie-afhandelingsroutines in het formulier moet zijn geladen. Het commando SET PROCEDURE...ADDITIVE wordt impliciet uitgevoerd voor de programmabestanden (ook .WFM-bestanden) waarmee formulieren wordt gemaakt, om te zorgen dat de procedures beschikbaar zijn. Op deze manier komen methoden, actie-afhandelingsroutines en andere procedures in het programmabestand beschikbaar voor uitvoering. De procedures blijven geladen totdat alle formulieren in het bestand zijn *vrijgegeven*.

Programmeerstrategieën

Zoals u eerder in dit hoofdstuk hebt gelezen, zijn de meeste formulieren in een actiegestuurde applicatie niet-modaal, zodat gebruikers kunnen schakelen naar andere formulieren, of zelfs naar het commandovenster of Navigator. Hoewel deze bewegingsvrijheid gunstig is voor gebruikers, moeten programmeurs de applicatie zorgvuldig ontwerpen om te zorgen dat deze kan worden gebruikt in combinatie met andere hulpmiddelen.

Dit kan een reden zijn om geen volledig actiegestuurde applicatie te schrijven. In de volgende secties worden drie strategieën besproken die u kunt hanteren. U kunt de strategieën apart of in combinatie toepassen.

Modale interface

Open alle formulieren als modale formulieren met de formuliermethode ReadModal() of de functie READMODAL(). Deze aanpak zal met name programmeurs met ervaring in dBASE III PLUS en dBASE IV aanspreken. Een modale gebruikersinterface heeft de volgende eigenschappen:

- *Elk formulier heeft exclusieve focus*, zodat gebruikers het formulier moeten sluiten voordat ze kunnen doorgaan. Hoewel dit beperkingen oplegt aan gebruikers, is deze aanpak eenvoudiger voor programmeurs, omdat het aantal mogelijke acties van gebruikers kleiner is.
- *De dBASE-interface is niet toegankelijk*, zolang de applicatie actief is. Op deze manier voorkomt u dat gebruikers tabellen bewerken die worden gebruikt in de applicatie, terwijl de applicatie wordt uitgevoerd.
- *De programma-uitvoering stopt wanneer elk formulier wordt weergegeven*, waardoor het bereik van variabelen en de beschikbaarheid van procedures gemakkelijker is te beheren.

Een modale interface is in zekere mate toch actiegestuurd, omdat u actie-afhandelingsroutines kunt toewijzen aan formulieren en stuurelementen. De actievolgorde wordt echter bepaald door de modale applicaties (net als bij menugestuurde applicaties).

Met de volgende code maakt u het invoerformulier in Afbeelding 13.1 (u maakt het formulier dit keer echter modaal).

```
DEFINE FORM KlBewerken FROM 12.5,24.5 TO 31.5,70.38;
   PROPERTY;
      EscExit .T.,;
      View "KLBEWERK.QBE",;
      Text "Klant bewerken",;
      Minimize .F.,;
      Maximize .F.,;
      MDI .F.                  && Vereist voor modale formulieren

DEFINE TEXT Tekst1 OF KlBewerken;
   PROPERTY;
      ColorNormal "N/W",;
      Text "&Naam",;
      Width         10.00,;
      Top          1.00,;
      Left          2.00,;
      Height          1.00
```

```
DEFINE ENTRYFIELD Invoer1 OF KlBewerken;
   PROPERTY;
      ColorNormal "",;
      Width         30.00,;
      Top          1.00,;
      Left         13.00,;
      Height          1.50,;
      Border .T.,;
      DataLink "KLANTEN->NAAM"

*** De TEXT- en ENTRYFIELD-definities voor de overige invoervakken worden op dezelfde
*** manier gecodeerd. Deze definities zijn weggelaten om ruimte te sparen.

DEFINE PUSHBUTTON Knop1 OF KlBewerken;
   PROPERTY;
      OnClick KNOP1_ONCLICK,;
      ColorNormal "",;
      Text "Zoeken",;
      Width         10.00,;
      Top         15.00,;
      Left         17.00,;
      Height          2.00

KlBewerken.ReadModal()            && Opent het formulier als modaal formulier

PROCEDURE Knop1_OnClick
   DO KLZOEK.PRG
RETURN
```

Met de volgende code maakt u het modale dialoogvenster in Afbeelding 13.1.

```
* KLZOEK.PRG
*
DEFINE FORM KlZoeken FROM 11,30 TO 18,75;
   PROPERTY;
      EscExit .T.,;
      Text "Naam zoeken",;
      Minimize .F.,;
      Maximize .F.,;
      MDI .F.                    && Vereist voor modale formulieren

DEFINE ENTRYFIELD Invoer1 OF KlZoeken;
   PROPERTY;
      ColorNormal "",;
      Width         26.00,;
      Top          1.00,;
      Left         18.00,;
      Height         1.00,;
      Border .T.,;
      Value "",;
      MaxLength 25
```

```
DEFINE TEXT Tekst1 OF KlZoeken;
   PROPERTY;
      ColorNormal "N/W",;
      Text "Naam",;
      Width         10.00,;
      Top          1.00,;
      Left          2.00,;
      Height          2.00

DEFINE PUSHBUTTON Knop1 OF KlZoeken;
   PROPERTY;
      OnClick Knop1_OnClick,;
      ColorNormal "",;
      Text "OK",;
      Width         10.00,;
      Top          5.00,;
      Left         18.00,;
      Height          2.00,;
      Default .T.

DEFINE PUSHBUTTON Knop2 OF KlZoeken;
   PROPERTY;
      OnClick {;CLOSE FORM KlZoeken},;
      ColorNormal "",;
      Text "Annuleren",;
      Width         15.00,;
      Top          5.00,;
      Left         30.00,;
      Height          2.00

READMODAL(KlZoeken)                    && Opent het formulier als modaal formulier

PROCEDURE Knop1_OnClick
   IF .NOT. EMPTY(KlZoeken.Invoer1.Value)
       LOCAL lExact
       lExact = SET("EXACT")
       SET EXACT off
       SEEK(UPPER(TRIM(KlZoeken.Invoer1.Value)))
       SET EXACT &lExact
   ENDIF
   CLOSE FORM KlZoeken
RETURN
```

Niet-modale interface

Open de meeste formulieren als niet-modale formulieren met de formuliermethode Open() of het commando OPEN FORM. Gebruik zo weinig mogelijk modale formulieren om benodigde informatie te verkrijgen van gebruikers. Deze aanpak is gunstig voor gebruikers en biedt de volgende voordelen:

- *Gebruikers kunnen meer dan één programma tegelijkertijd starten.* In een grote applicatie kunnen gebruikers bijvoorbeeld een invoerformulier voor orders starten, dit

formulier voor de helft invullen, een inventarisrapport afdrukken en vervolgens teruggaan naar het formulier om dit verder in te vullen. Deze werkwijze sluit aan op de werkzaamheden van alledag.

- *Gebruikers kunnen toegang krijgen tot de dBASE-interface,* bijvoorbeeld het commandovenster en Navigator. Dit is zeer handig wanneer u hulpprogramma's schrijft met speciale opzoek- of weergavefuncties die een aanvulling vormen op de dBASE-interface. U kunt de dBASE-interface desgewenst uitschakelen met de functie SHELL().
- *De programma-uitvoering gaat door tot aan het einde, en stopt zelfs niet wanneer formulieren worden weergegeven.* De rest van het programma wordt geïnitieerd door actie-afhandelingsroutines die worden uitgevoerd als gevolg van gebruikersacties.

Een niet-modale interface is volledig actiegestuurd. Wanneer het beginscherm eenmaal wordt weergegeven, bepalen de gebruikers de actievolgorde.

Met de volgende code maakt u het invoerformulier in Afbeelding 13.1.

```
* KLBEWERK.PRG
*
*

DEFINE FORM KlBewerken FROM 12.5,24.5 TO 31.5,70.38;
   PROPERTY;
      EscExit .T.,;
      View "KLBEWERK.QBE",;
      Text "Klant bewerken",;
      Minimize .F.,;
      Maximize .F.

DEFINE TEXT Tekst1 OF KlBewerken;
   PROPERTY;
      ColorNormal "N/W",;
      Text "&Naam",;
      Width         10.00,;
      Top          1.00,;
      Left          2.00,;
      Height          1.00

DEFINE ENTRYFIELD Invoer1 OF KlBewerken;
   PROPERTY;
      ColorNormal "",;
      Width         30.00,;
      Top          1.00,;
      Left         13.00,;
      Height          1.50,;
      Border .T.,;
      DataLink "KLANTEN->NAAM"

*** De TEXT- en ENTRYFIELD-definities voor de overige invoervakken worden op dezelfde
*** manier gecodeerd. Deze definities zijn weggelaten om ruimte te sparen.
```

```
DEFINE PUSHBUTTON Knop1 OF KlBewerken;
   PROPERTY;
      OnClick Knop1_OnClick,;
      ColorNormal "",;
      Text "Zoeken",;
      Width         10.00,;
      Top          15.00,;
      Left          17.00,;
      Height          2.00

OPEN FORM KlBewerken                  && Opent het formulier als niet-modaal formulier

PROCEDURE Knop1_OnClick
   DO KLZOEK.PRG
RETURN
```

Met de volgende code maakt u het modale dialoogvenster in Afbeelding 13.1.

```
* KLZOEK.PRG
*
DEFINE FORM KlZoeken FROM 11,30 TO 18,75;
   PROPERTY;
      EscExit .T.,;
      Text "Naam zoeken",;
      Minimize .F.,;
      Maximize .F.,;
      MDI .F.                  && Vereist voor modale formulieren

DEFINE ENTRYFIELD Invoer1 OF KlZoeken;
   PROPERTY;
      ColorNormal "",;
      Width         26.00,;
      Top          1.00,;
      Left         18.00,;
      Height          2.00,;
      Border .T.,;
      Value "",;
      MaxLength 25

DEFINE TEXT Tekst1 OF KlZoeken;
   PROPERTY;
      ColorNormal "N/W",;
      Text "Naam",;
      Width         10.00,;
      Top          1.00,;
      Left          2.00,;
      Height          2.00
```

```
DEFINE PUSHBUTTON Knop1 OF KlZoeken;
   PROPERTY;
      OnClick Knop1_OnClick,;
      ColorNormal "",;
      Text "OK",;
      Width         10.00,;
      Top          5.00,;
      Left         18.00,;
      Height          2.00,;
      Default .T.

DEFINE PUSHBUTTON Knop2 OF KlZoeken;
   PROPERTY;
      OnClick {;CLOSE FORM KlZoeken},;
      ColorNormal "",;
      Text "Annuleren",;
      Width         15.00,;
      Top          5.00,;
      Left         30.00,;
      Height          2.00

KlZoeken.ReadModal()                    && Opent het formulier als modaal formulier

PROCEDURE Knop1_OnClick
   IF .NOT. EMPTY(KlZoeken.Invoer1.Value)
       PRIVATE lExact
       lExact = SET("EXACT")
       SET EXACT off
       SEEK(UPPER(TRIM(KlZoeken.Invoer1.Value)))
       SET EXACT &lExact
   ENDIF
   CLOSE FORM KlZoeken
RETURN
```

Object-georiënteerde, niet-modale interface

Maak formulieren door eigen klassen te definiëren en deze klassen weer te geven als niet-modale formulieren. Formulierontwerp genereert object-georiënteerde code. Deze aanpak vormt de krachtigste manier om een niet-modale, actiegestuurde interface te implementeren.

Zie Hoofdstukken 9 tot en met 12 voor meer informatie over object-georiënteerd programmeren.

Met de volgende code, die wordt gegenereerd door Formulierontwerp, definieert u een klasse om het invoerformulier in Afbeelding 13.1 te maken.

```
* KLBEWERK.WFM
* END HEADER - deze regel niet verwijderen
* gemaakt op 30-09-94
*
LOCAL f
f = NEW KLBEWERK ()
f.Open()
CLASS KLBEWERKEN OF FORM
   this.EscExit = .T.
   this.View = "KLBEWERK.QBE"
   this.Text = "Klant bewerken"
   this.Width =         45.88
   this.Top =         12.50
   this.Left =         24.50
   this.Height =         19.00
   this.Minimize = .F.
   this.Maximize = .F.
    DEFINE TEXT Tekst1 OF THIS;
       PROPERTY;
          ColorNormal "N/W",;
          Text "&Naam",;
          Width         10.00,;
          Top          1.00,;
          Left          2.00,;
          Height          1.00
    DEFINE ENTRYFIELD Invoer1 OF THIS;
       PROPERTY;
          ColorNormal "",;
          Width         30.00,;
          Top          1.00,;
          Left         13.00,;
          Height          1.50,;
          Border .T.,;
          DataLink "KLANTEN->NAAM"

*** De TEXT- en ENTRYFIELD-definities voor de overige invoervakken worden op dezelfde
*** manier gecodeerd. Deze definities zijn weggelaten om ruimte te sparen.

    DEFINE PUSHBUTTON Knop1 OF THIS;
       PROPERTY;
          OnClick CLASS::Knop1_OnClick,;
          ColorNormal "",;
          Text "Zoeken",;
          Width         10.00,;
          Top         15.00,;
          Left         17.00,;
          Height          2.00

   PROCEDURE Knop1_OnClick
      DO KLZOEK.WFM
   RETURN
```

```
ENDCLASS
```

Met de volgende code, die wordt gegenereerd door Formulierontwerp, definieert u een klasse om het modale dialoogvenster in Afbeelding 13.1 te maken.

```
* KLZOEK.WFM
* END HEADER - deze regel niet verwijderen
* gemaakt op 30-09-94
*
LOCAL f
f = NEW KLZOEKFORM()
f.ReadModal()

CLASS KLZOEKFORM OF FORM
   this.EscExit = .T.
   this.Text = "Naam zoeken"
   this.Width =         45.00
   this.Top =          11.00
   this.Left =          30.00
   this.Height =           7.00
   this.Minimize = .F.
   this.Maximize = .F.
   this.MDI = .F.

    DEFINE ENTRYFIELD Invoer1 OF THIS;
       PROPERTY;
          ColorNormal "",;
          Width          26.00,;
          Top           1.00,;
          Left          18.00,;
          Height           1.00,;
          Border .T.,;
          Value "",;
          MaxLength 25

    DEFINE TEXT Tekst1 OF THIS;
       PROPERTY;
          ColorNormal "N/W",;
          Text "Naam",;
          Width          15.00,;
          Top           1.00,;
          Left           2.00,;
          Height           2.00

    DEFINE PUSHBUTTON Knop1 OF THIS;
       PROPERTY;
          OnClick CLASS::Knop1_OnClick,;
          ColorNormal "",;
          Text "OK",;
          Width          10.00,;
          Top           5.00,;
          Left          18.00,;
          Height           2.00,;
```

```
        Default .T.
    DEFINE PUSHBUTTON Knop2 OF THIS;
       PROPERTY;
          OnClick {form.Close()},;
          ColorNormal "",;
          Text "Annuleren",;
          Width          15.00,;
          Top            3.00,;
          Left           30.00,;
          Height          2.00

    PROCEDURE Knop1_OnClick
       IF .NOT. EMPTY(form.Invoer1.Value)
          PRIVATE lExact
          lExact = SET("EXACT")
          SET EXACT OFF
          SEEK(UPPER(TRIM(form.Invoer1.Value)))
          SET EXACT &lExact
       ENDIF
       form.Close()
    RETURN

ENDCLASS
```

Werking van de Enter-toets instellen

Een van de verschillen tussen dBASE III+/IV en dBASE voor Windows is de werking van *Enter*. Als u op *Enter* drukt in dBASE III+/IV, gaat de cursor naar het volgende commando GET, en bij het laatste commando GET wordt het commando READ beëindigd. In Windows-toepassingen wordt meestal *Tab* gebruikt om de focus te verplaatsen naar het volgende object in een venster. Als u op *Enter* drukt, wordt er een formulier voorgelegd, net als met *Ctrl+End* (of *Ctrl+W*) in dBASE III+/IV.

Als u op *Enter* drukt in dBASE wanneer het formuliervenster (maar geen editor-stuurelement) de focus heeft, wordt het formulier voorgelegd. (Zie Hoofdstuk 14 voor meer informatie over het voorleggen van een formulier.) Met *Tab* en *Shift+Tab* wordt de focus verplaatst naar het volgende en vorige object.

Hoewel de Windows-werking van *Enter* de voorkeur heeft in een Windows-applicatie, kunt u ervoor kiezen de DOS-werking in te stellen om consistent te blijven met uw DOS-applicaties. U geeft als volgt de werking van *Enter* op met het commando SET CUAENTER:

- SET CUAENTER OFF. Hiermee werkt *Enter* als in dBASE III+/IV (de focus wordt verplaatst naar het volgende object).
- SET CUAENTER ON. Hiermee werkt *Enter* als in Windows (het formulier wordt voorgelegd).

Aanvullende informatie

Een algemeen erkend model voor actiegestuurde gebruikersinterfaces zijn de CUA-richtlijnen (Common User Access) van IBM. Veel van deze richtlijnen worden toegepast in de Windows-omgeving, in een aangepaste vorm. De objecten van de gebruikersinterface in de dBASE-taal zijn ontworpen op basis van CUA, zodat u gebruikersinterfaces kunt maken die de CUA-richtlijnen volgen.

Als u meer wilt weten over hoe u effectieve actiegestuurde gebruikersinterfaces kunt maken, kunt u de volgende boeken raadplegen:

- *Object-oriented Interface Design: IBM Common User Access Guidelines*, uitgegeven door IBM. Dit boek introduceert de CUA-concepten en is tevens een naslaggids met richtlijnen voor elk element van de gebruikersinterface
- *The Windows Interface: An Application Design Guide*, uitgegeven door Microsoft Press.

Hoofdstuk

14

Acties afhandelen

Wanneer u programmeert met formulieren, schrijft u geen code die op volgorde wordt uitgevoerd. In plaats daarvan schrijft u *actie-afhandelingsprocedures*, die u koppelt aan het formulier en/of de bijbehorende stuurelementen. Actie-afhandelingsroutines worden uitgevoerd wanneer de corresponderende acties plaatsvinden.

In dit hoofdstuk wordt beschreven hoe u actie-afhandelingsroutines schrijft en in welke volgorde de actie-afhandelingsroutines worden uitgevoerd.

Zie Hoofdstuk 2 voor een introductie van actiegestuurde programma's.

Werken met actie-afhandelingsroutines

Een actie-afhandelingsroutine is een procedure, functie of codeblok. De code in een actie-afhandelingsroutine volgt de normale regels voor programmaverloop: commando's worden regel voor regel uitgevoerd, en voorwaardelijke opdrachten zoals DO WHILE...ENDDO of IF...ENDIF kunnen sprongen of herhalingen instellen. Het verschil tussen actie-afhandelingsroutines en traditionele procedures is de context waarin ze worden uitgevoerd.

In een traditionele menugestuurde applicatie worden procedures aangeroepen op vaste punten in de programmacode. Hierbij kunnen programmeurs gemakkelijker bepalen in welke omgeving de procedure wordt uitgevoerd (zodat ze bijvoorbeeld weten welke geheugenvariabelen beschikbaar zijn), omdat ze kunnen zien welke code wordt uitgevoerd vóór de procedure.

In een actiegestuurde applicatie wijzen programmeurs actie-afhandelingsroutines toe aan acties en worden de procedures uitgevoerd wanneer de actie plaatsvindt. Hierbij is er weinig bekend over de beschikbaarheid van geheugenvariabelen, omdat de procedure heel algemeen moet zijn en moet kunnen worden uitgevoerd in elke willekeurige programma-omgeving.

De beste manier om actie-afhandelingsroutines te schrijven is algemene termen (zoals *this* en *form*) te gebruiken om te verwijzen naar de objecten waarop de code betrekking

heeft (zie volgend voorbeeld). Zie Hoofdstuk 11 voor meer informatie over het gebruik van *this* en *form* om te verwijzen naar objectleden.

Met de volgende code maakt u een formulier met een knop en wijst u een actie-afhandelingsroutine (KlikMelding) toe aan de actie OnClick van de knop.

```
DEFINE FORM ActieTest
DEFINE PUSHBUTTON MijnKnop OF ActieTest
ActieTest.MijnKnop.OnClick = KlikMelding    && Actie-afhandelingsroutine toewijzen aan
                                            && actie OnClick
```

In Afbeelding 14.1 ziet u de bijbehorende actie-afhandelingsroutines KlikMelding. De routine aan de linkerkant heeft vast gecodeerde verwijzingen en de code aan de rechterkant heeft algemene verwijzingen:

Afbeelding 14.1 Voorbeelden van vast gecodeerde en algemene actieafhandelingsroutines

Actie-afhandelingsroutine met gecodeerde verwijzingen

```
PROCEDURE KlikMelding

  * Venstertitel wijzigen
  ActieTest.Text = "Hé! Je drukte op mijn knop!"

  * Knoplabel wijzigen
  ActieTest.MijnKnop.Text = "Hmpf!"
RETURN
```

Actie-afhandelingsroutine met algemene verwijzingen

```
PROCEDURE KlikMelding

  * Venstertitel wijzigen
  form.Text = "Hé! Je drukte op mijn knop!"

  * Knoplabel wijzigen
  this.Text = "Hmpf!"
RETURN
```

Beide codevoorbeelden voeren dezelfde taak uit, maar de linkercode werkt alleen wanneer deze wordt toegewezen aan de knop MijnKnop in het formulier ActieTest. De rechtercode werkt bij elke willekeurige knop in elk willekeurig formulier.

Belangrijk De verwijzingen *this* en *form* zijn alleen beschikbaar voor procedures die rechtstreeks zijn gekoppeld aan actiekenmerken, en niet voor procedures die later worden aangeroepen. Als een actie-afhandelingsroutine een andere procedure aanroept en de tweede procedure maakt gebruikt van *this* of *form*, moet u *this* of *form* overbrengen als parameter(s) naar de tweede procedure.

In het volgende voorbeeld wordt de code aan de rechterkant in Afbeelding 14.1 gewijzigd, waarbij *form* wordt overgebracht naar een tweede procedure.

```
PROCEDURE KlikMelding
   form.Text = "Hé! Je drukte op mijn knop!"   && Venstertitel wijzigen
   this.Text = "Hmpf!"                         && Knoplabel wijzigen
   DO Terugkaatsen WITH form
RETURN

PROCEDURE Terugkaatsen(FormRef)
      FormRef.ColorNormal = "r/w*"
      SLEEP 2
      FormRef.ColorNormal = "r/w"
RETURN
```

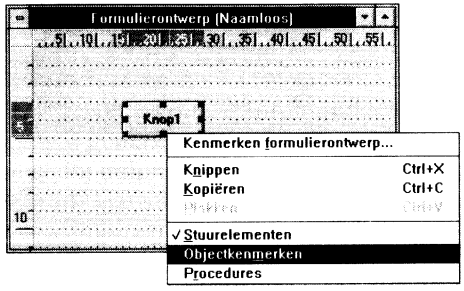
Actie-afhandelingsroutines schrijven met Formulierontwerp

In de voorbeelden in dit hoofdstuk ligt de nadruk op actie-afhandelingscode. Wanneer u formulieren maakt, gebruikt u waarschijnlijk Formulierontwerp en de bijbehorende hulpmiddelen om de code te schrijven. In het volgende voorbeeld leert u stapsgewijs hoe u actie-afhandelingsroutines schrijft en koppelt met Formulierontwerp. U maakt twee actie-afhandelingsroutinesvoor een knop die bijhoudt hoe vaak erop wordt geklikt. (U moet bekend zijn met Formulierontwerp en de bijbehorende hulpmiddelen om dit voorbeeld te begrijpen. Zie Hoofdstuk 8 in het *Handboek* voor meer informatie over Formulierontwerp en de bijbehorende hulpmiddelen.)

1 Start Formulierontwerp en plaats een knop op een leeg formulier.

2 Rechtsklik op de knop en kies **Objectkenmerken** in het snelmenu om het bijbehorende kenmerkenvenster weer te geven.

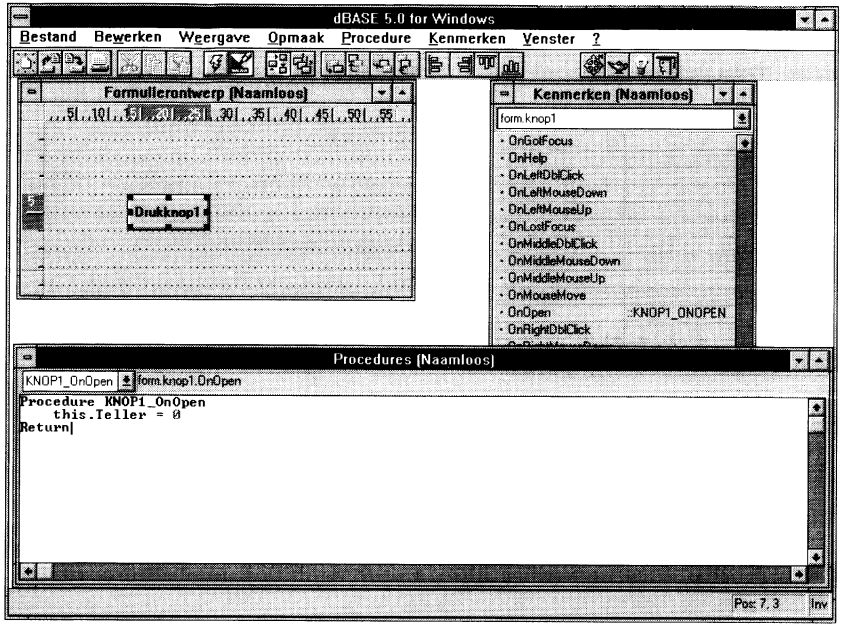
Afbeelding 14.2 Het kenmerkenvenster openen



3 Selecteer de actie **OnOpen** van de knop en klik op de hulpmiddelenknop. De procedure-editor verschijnt.

4 Typ de korte procedure in Afbeelding 14.3:

Afbeelding 14.3 Een afhandelingsroutine voor de actie OnOpen van een knop



Met de regel `this.Teller = 0` wordt een eigen kenmerk van de knop geïnitialiseerd. Omdat de afhandelingsroutine voor de actie **OnOpen** is gekoppeld aan de knop, verwijst *this* naar het knopobject. Wanneer u het formulier opent, wordt de procedure **KNOP1_OnOpen** uitgevoerd en wordt het kenmerk **Teller** van de knop gemaakt.

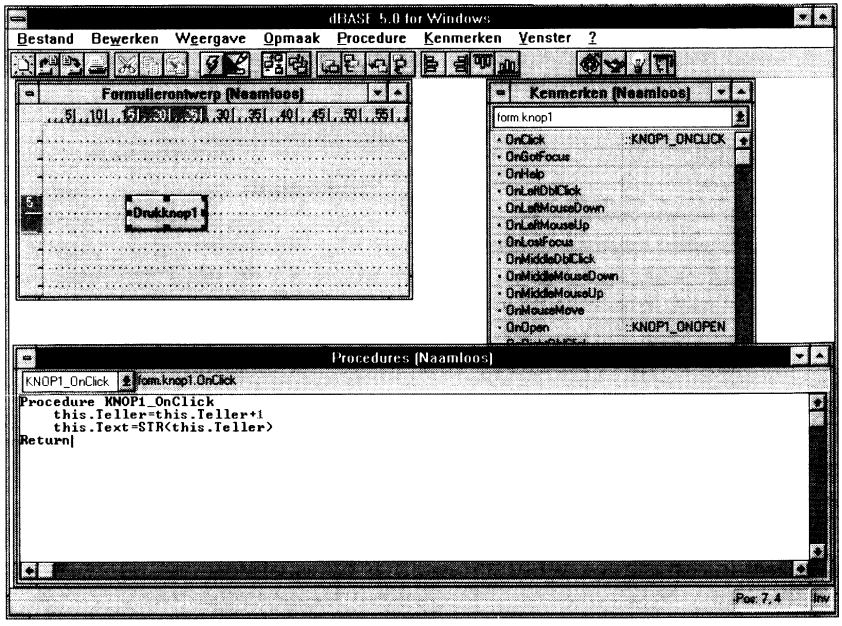
5 Sluit het venster van de procedure-editor.

U hebt zojuist een actie-afhandelingsroutine gemaakt die is gekoppeld aan de actie **OnOpen** van de knop. U gaat nu een actie-afhandelingsroutine maken voor de actie **OnClick**.

6 Selecteer de actie **OnClick** in het kenmerkenvenster van de knop en klik op de hulpmiddelenknop. Er verschijnt een lege procedure in de procedure-editor.

7 Typ de korte procedure in Afbeelding 14.4:

Afbeelding 14.4 Een afhandelingsroutine voor de actie OnClick van een knop



Met de eerste regel van de procedure **KNOP1_OnClick** wordt het kenmerk **Teller** stapsgewijs verhoogd. De tweede regel verwijst naar het kenmerk **Text** en slaat hierin de waarde van **Teller** op die wordt geconverteerd naar een tekenreeks. Het kenmerk **Text** verschijnt als label op de knop.

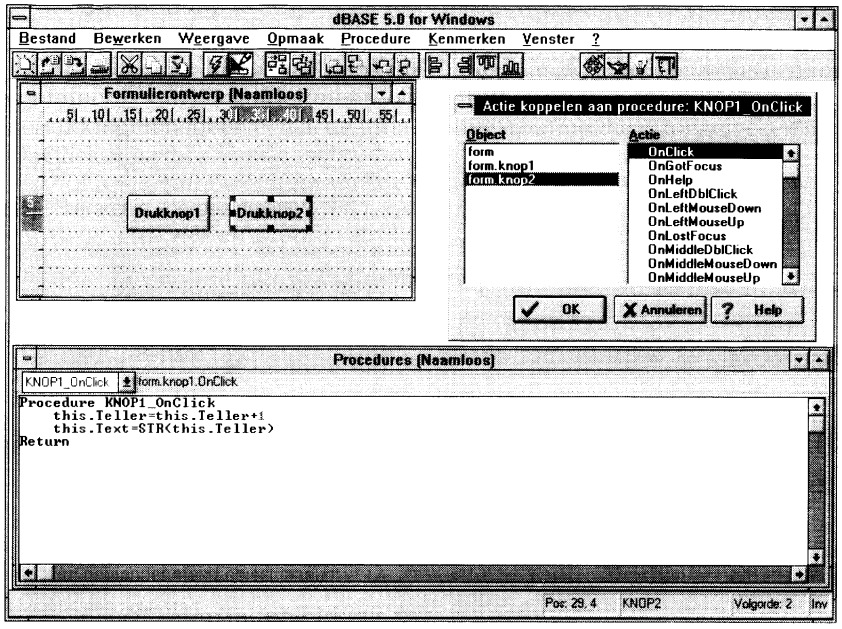
8 Laat de procedure-editor open en plaats een tweede knop in het formulier.

Als u de tweede knop eveneens het aantal klikhandelingen wilt laten bijhouden, kunt u de actie-afhandelingsroutines van de eerste knop koppelen aan de tweede knop zonder opnieuw code in te voeren.

9 Rechtsklik in het venster van de procedure-editor om het bijbehorende snelmenu weer te geven, en kies vervolgens **Actie koppelen** om het dialoogvenster **Actie koppelen aan procedure** weer te geven.

10 Selecteer **form.knop2** in de keuzelijst **Object** en selecteer **OnClick** in de keuzelijst **Actie** (zie Afbeelding 14.5).

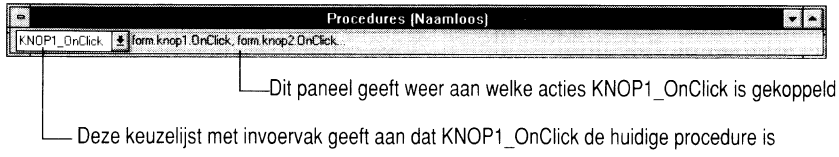
Afbeelding 14.5 Een actie-afhandelingsroutine koppelen aan een andere actie



In het dialoogvenster **Actie koppelen aan procedure** koppelt u de huidige actie-afhandelingsroutine in de procedure-editor aan de geselecteerde actie van het geselecteerde object.

11 Klik op **OK** om de actie te koppelen. In het paneel aan de bovenkant van de procedure-editor (zie Afbeelding 14.6) wordt aangegeven dat de procedure is gekoppeld aan **form.knop1.OnClick** en **form.knop2.Onclick**.

Afbeelding 14.6 Een actie-afhandelingsroutine die is gekoppeld aan twee acties

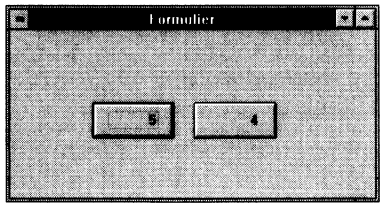


12 Selecteer de procedure **KNOP1_OnOpen** in de keuzelijst met invoervak (zie Afbeelding 14.6) en koppel deze procedure op dezelfde manier aan de actie **OnOpen** van de tweede knop.

13 Sla het formulier op, schakel naar de uitvoermodus en klik een paar keer op elke knop.

Zoals u ziet, houdt elke knop een eigen teller bij.

Afbeelding 14.7 Het voorbeeldformulier met twee knoppen



U leert het volgende van dit voorbeeld:

- Als u de variabele *Teller* definieert als kenmerk van het knopobject, wordt de variabele ingesloten en zo beschermd tegen overschrijven. Beide knoppen hebben een kenmerk **Teller**, maar de kenmerken worden onafhankelijk van elkaar bijgewerkt.
- Als u *this* gebruikt om te verwijzen naar kenmerken, maakt u actie-afhandelingsroutines algemeen en geschikt om opnieuw te gebruiken. De uitdrukkingen `this.Teller` en `this.Text` verwijzen naar de kenmerken **Teller** en **Text** van de objecten waaraan de actie-afhandelingsroutines zijn gekoppeld. Als u de objectverwijzing codeert (met `knop1.Teller` in plaats van `this.Teller`), kunt u de actie-afhandelingsroutine niet gemakkelijk koppelen aan een andere knop.

Leren wanneer acties plaatsvinden

Niet alle acties worden rechtstreeks gestart door gebruikers. Sommige acties worden door andere acties geïnitieerd. Als u bijvoorbeeld klikt op een knop die niet de focus heeft, probeert u de focus te verplaatsen naar deze knop. Voordat de afhandelingsroutine voor de actie OnClick van de knop wordt uitgevoerd, vinden de volgende acties plaats:

1. De afhandelingsroutine voor de actie When van de knop wordt uitgevoerd om te bepalen of de knop de focus kan krijgen
2. Als de knop de focus kan krijgen, wordt de afhandelingsroutine voor de actie OnGotFocus uitgevoerd
3. Nu wordt de afhandelingsroutine voor de actie OnClick van de knop uitgevoerd

De volgorde waarin de actie-afhandelingsroutines worden uitgevoerd, wordt nog ingewikkelder als een ander stuurelement al de focus heeft wanneer u klikt op een knop. Het stuurelement dat de focus verliest, kan nog meer actie-afhandelingsroutines hebben (bijvoorbeeld Valid en OnLostFocus) die worden opgenomen in de uitvoervolgorde.

Als u vijf of meer actie-afhandelingsroutines kunt starten met één enkele klikhandeling, hoe kunt u dan bepalen welke actie-afhandelingsroutines u moet toewijzen aan elk stuurelement? Dit is gelukkig gemakkelijker dan het lijkt. U hoeft niet bij voorbaat *alle* acties te overwegen die kunnen plaatsvinden wanneer een formulier actief is, alleen de acties die betrekking hebben op uw formulier. Als u geen code toewijst aan de actie-

afhandelingsroutine voor een kenmerk, wordt dit kenmerk overgeslagen en wordt de volgende actie-afhandelingsroutine uitgevoerd, totdat alle acties zijn verwerkt.

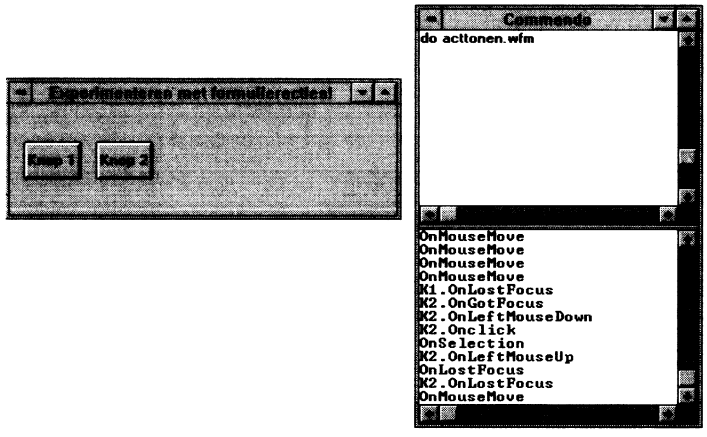
In het volgende voorbeeldprogramma experimenteert u met diverse actie-afhandelingsroutines en leert u wanneer deze routines worden uitgevoerd. U wijst een codeblok toe aan elke actie, waarmee een melding wordt weergegeven in het commandovenster om aan te geven dat de actie heeft plaatsgevonden. Wanneer u het formulier uitvoert, kunt u zien in het commandovenster welke acties plaatsvinden terwijl u van de ene knop naar de andere gaat.

```
* ACTTONEN.WFM
LOCAL f
f = NEW ACTTONEN()
f.Open()
CLASS ACTTONEN OF FORM
   this.OnSize = {;? "OnSize"}
   this.OnMouseMove = {;? "OnMouseMove"}
   this.OnRightMouseUp = {;? "OnRightMouseUp"}
   this.OnRightMouseDown = {;? "OnRightMouseDown"}
   this.OnLeftMouseUp = {;? "OnLeftMouseUp"}
   this.OnLeftMouseDown = {;? "OnLeftMouseDown"}
   this.OnLostFocus = {;? "OnLostFocus"}
   this.OnGotFocus = {;? "OnGotFocus"}
   this.OnOpen = {;? "OnOpen  "}
   this.OnSelection = {;? "OnSelection"}
   this.OnClose = {;? "OnClose "}
   this.OnMove = {;? "OnMove"}
   this.Text = "Experimenteren met formulieracties!"
   this.Width =          45.00
   this.Height =         8.06
   this.Top =          7.94
   this.Left =          54.20
   DEFINE PUSHBUTTON K1 OF THIS;
       PROPERTY;
          OnRightMouseUp {;? "K1.OnRightMouseUp"},;
          OnRightMouseDown {;? "K1.OnRightMouseDown"},;
          OnLeftMouseUp {;? "K1.OnLeftMouseUp"},;
          OnLeftMouseDown {;? "K1.OnLeftMouseDown"},;
          OnLostFocus {;? "K1.OnLostFocus"},;
          OnGotFocus {;? "K1.OnGotFocus"},;
          OnClick {;? "K1.Onclick"},;
          Text "Knop 1",;
          Width          14.00,;
          Height          2.00,;
          Top          3.00,;
          Left          4.00,;
          Group .T.
```

```
    DEFINE PUSHBUTTON K2 OF THIS;
       PROPERTY;
          OnRightMouseUp {;? "K2.OnRightMouseUp"},;
          OnRightMouseDown {;? "K2.OnRightMouseDown"},;
          OnLeftMouseUp {;? "K2.OnLeftMouseUp"},;
          OnLeftMouseDown {;? "K2.OnLeftMouseDown"},;
          OnLostFocus {;? "K2.OnLostFocus"},;
          OnGotFocus {;? "K2.OnGotFocus"},;
          OnClick {;? "K2.Onclick"},;
          Text "Knop 2",;
          Width          13.00,;
          Height          2.00,;
          Top          3.00,;
          Left         22.00,;
          Group .T.
ENDCLASS
```

In Afbeelding 14.8 ziet u het formulier ActTonen.

Afbeelding 14.8 ACTTONEN.WFM



Code schrijven voor initialisatie- en herstelprocedures

Voor sommige formulieren, met name formulieren die zijn gebaseerd op tabellen, moet u initialisatiecode schrijven om de omgeving in te stellen wanneer het formulier wordt geopend. Formulieren kunnen ook herstelcode vereisen die moet worden uitgevoerd wanneer het formulier wordt gesloten. Gebruik de formulieractie OnOpen om initialisatiecode toe te wijzen en de formulieractie OnClose om herstelcode toe te wijzen. Daarnaast heeft ook elk standaardstuurelement in het formulier de actie OnOpen, die wordt uitgevoerd wanneer het formulier wordt geopend. Als u code toewijst aan de actie OnOpen van elk stuurelement, kunt u het formulier stuurelement voor stuurelement initialiseren.

- Afhandelingsroutines voor de actie OnOpen voeren initialisatieprocedures uit, zoals geheugenvariabelen definiëren of tabellen openen die niet zijn opgegeven bij het formulierkenmerk View.
- Afhandelingsroutines voor de actie OnClose voeren herstelprocedures uit, zoals het formulier vrijgeven of geheugenvariabelen vrijgeven.

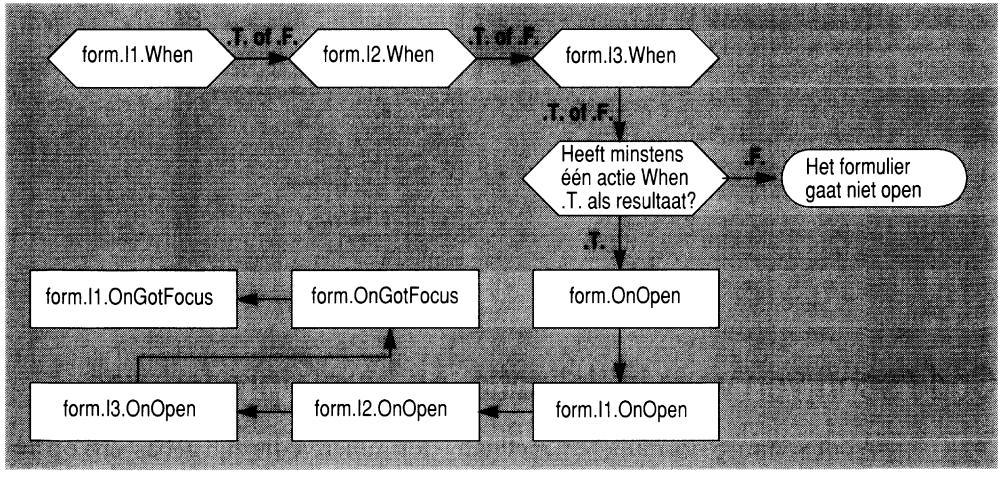
Wanneer u formulieren maakt met Formulierontwerp, kunt u ook initialisatiecode opgeven in de header van het .WFM-bestand. Code in de header wordt uitgevoerd wanneer u het .WFM-bestand start, in plaats van wanneer u het formulier opent . Zie Hoofdstuk 16 voor meer informatie over het invoeren van code in de header van een .WFM-bestand.

Er kunnen veel acties worden uitgevoerd bij het openen en sluiten van formulieren. In de volgende twee secties ziet u een uitgebreid overzicht van de actievolgorde.

Actievolgorde bij het openen van een formulier

Wanneer u een formulier opent, krijgt het formuliervenster de focus. Bovendien wordt er nagegaan in hoeverre alle stuurelementen in het formulier de focus kunnen krijgen, en wordt de focus toegewezen aan het eerste stuurelement dat in aanmerking komt. In Afbeelding 14.9 ziet u een overzicht van de volgorde waarin actie-afhandelingsroutines worden uitgevoerd wanneer u een formulier met drie invoervakken (I1, I2 en I3) opent.

Afbeelding 14.9 De actievolgorde bij het openen van een formulier met drie stuurelementen



Als u een formulier opent en u hebt nog geen andere formulieren geopend, gebeurt het volgende:

1 De afhandelingsroutines voor de actie When van alle stuurelementen in het formulier worden uitgevoerd. Als minstens één van de routines resulteert in .T. (waar), gaat de uitvoering door bij stap 2. Zo niet, dan gaat het formulier niet open.

2 De afhandelingsroutine voor de actie OnOpen van het formulier wordt uitgevoerd. Het venster wordt geopend en krijgt de focus.

3 De afhandelingsroutines voor de actie OnOpen van alle stuurelementen in het formulier worden uitgevoerd.

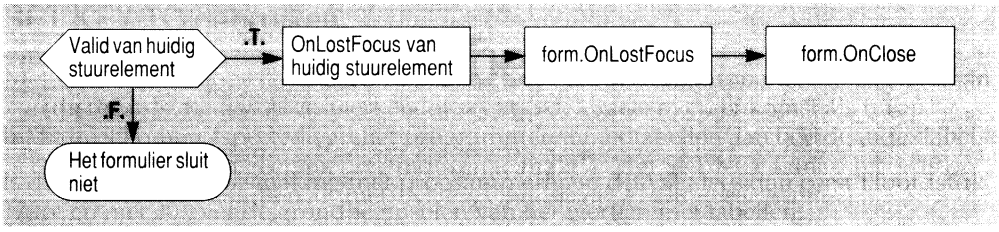
4 De afhandelingsroutine voor de actie OnGotFocus van het formulier wordt uitgevoerd. Het eerste stuurelement in de tabvolgorde krijgt de focus.

5 De afhandelingsroutine voor de actie OnGotFocus van het eerste stuurelement in de tabvolgorde wordt uitgevoerd.

Actievolgorde bij het sluiten van een formulier

Wanneer u een formulier sluit, verliezen het formuliervenster en het huidige stuurelement de focus. Bovendien verdwijnt het formulier van het scherm. In Afbeelding 14.10 ziet u een overzicht van de volgorde waarin actie-afhandelingsroutines worden uitgevoerd wanneer u een formuliersluit.

Afbeelding 14.10 De actievolgorde bij het sluiten van een formulier



Wanneer u een formulier sluit, gebeurt het volgende:

1 De afhandelingsroutine voor de actie Valid van het huidige stuurelement wordt uitgevoerd. Als deze routine resulteert in .F. (onwaar), wordt het formulier niet gesloten.

2 De afhandelingsroutine voor de actie OnLostFocus van het huidige stuurelement wordt uitgevoerd.

3 Als het formulier de focus heeft, wordt de afhandelingsroutine voor de actie OnLostFocus van het formulier uitgevoerd.

4 De afhandelingsroutine voor de actie OnClose van het formulier wordt uitgevoerd.

Procedures starten

De meest gangbare actie-afhandelingsroutine is een afhandelingsroutine die een actie uitvoert naar aanleiding van een klikhandeling. U koppelt deze actie-afhandelingsroutines aan de actie OnClick van knoppen of menu-opties. Afhandelingsroutines voor de actie OnClick kunnen willekeurige procedures zijn, hoewel de meeste van deze afhandelingsroutines eenvoudige acties uitvoeren, zoals het huidige formulier sluiten of een nieuw formulier openen.

U kunt de volgende afhandelingsroutine voor de actie OnClick bijvoorbeeld koppelen aan de knop **OK** of de opdracht **Bestand | Sluiten**:

```
PROCEDURE FormSluiten
   form.Close()
   form.Release()
RETURN
```

U kunt de actie OnClik in formulieren afhandelen op de volgende manieren:

- U wijst subroutines direct toe aan het kenmerk OnClick van menu-opties en knoppen. Bij de meeste formulieren heeft deze methode de voorkeur.
- U handelt alle afhandelingsroutines voor de actie OnClick van het gehele formulier af met het commando ON SELECTION FORM of de actie OnSelection. Deze methode is geschikt voor programmeurs die ervaring hebben met de menutechnieken van dBASE DOS of de programmeertechnieken van Windows.

Wanneer u klikt op een knop of een menu-optie kiest, wordt het formulier voorgelegd en gebeurt het volgende:

- Als het formulier een of meer knoppen bevat en de geselecteerde knop (of de standaardknop als u geen knop hebt geselecteerd) heeft een afhandelingsroutine voor de actie OnClick, wordt deze afhandelingsroutine uitgevoerd.
- De afhandelingsroutine voor de actie OnSelection van het formulier of de subroutine ON SELECTION wordt uitgevoerd (de routine die u hebt opgegeven).

De waarde van het kenmerk Id voor de huidige of standaardknop of de geselecteerde menu-optie wordt overgebracht naar de afhandelingsroutine voor de actie OnSelection of de procedure ON SELECTION.

In het volgende voorbeeld ziet u een klassedefinitie voor een formulier met vier knoppen. Als de gebruiker op een knop drukt, wordt deze uitgeschakeld. In de afhandelingsroutine van de actie OnSelection wordt de parameter Id gebruikt om te bepalen welke knop wordt uitgeschakeld.

```
LOCAL f
f = NEW KNOPKIESFORM()
f.Open()

CLASS KNOPKIESFORM OF FORM
   this.Left =         30.40
   this.Height =         14.12
   this.HelpId = ""
   this.HelpFile = ""
   this.Text = "De werking van OnSelection"
   this.OnSelection = CLASS::FORM_ONSELECTION
   this.Maximize = .F.
   this.Width =         50.40
   this.Minimize = .F.
   this.Top =          6.18
```

```
DEFINE TEXT Tekst1 OF THIS;
    PROPERTY;
      Left          6.00,;
      Height          1.00,;
      Text "Druk op een knop om deze uit te schakelen",;
      ColorNormal "N/W",;
      Border .F.,;
      Width         42.00,;
      Top          1.00
DEFINE PUSHBUTTON Knop1 OF THIS;
    PROPERTY;
      Default = .T.,;
      Left          4.00,;
      Height          2.00,;
      Text "Knop",;
      ColorNormal "N/W",;
      Group .T.,;
      ID          1,;
      Width         17.00,;
      Top          4.00
DEFINE PUSHBUTTON Knop2 OF THIS;
    PROPERTY;
      Left          30.00,;
      Height          2.00,;
      Text "Knop",;
      ColorNormal "N/W",;
      Group .T.,;
      ID          2,;
      Width         17.00,;
      Top          4.00
DEFINE PUSHBUTTON Knop3 OF THIS;
    PROPERTY;
      Left          4.00,;
      Height          2.00,;
      Text "Knop",;
      ColorNormal "N/W",;
      Group .T.,;
      ID          3,;
      Width         17.00,;
      Top          8.00
DEFINE PUSHBUTTON Knop4 OF THIS;
    PROPERTY;
      Left          30.00,;
      Height          2.00,;
      Text "Knop",;
      ColorNormal "N/W",;
      Group .T.,;
      ID          4,;
      Width         17.00,;
      Top          8.00
```

```
PROCEDURE Form_OnSelection(stuurelementId)        && Parameter Id ontvangen
                                                  && als stuurelementId
   DO CASE
      CASE stuurelementId = 1
           form.knop1.colornormal = 'w/gb'
           form.knop1.enabled = .f.
           form.knop1.before.setfocus()
      CASE stuurelementId = 2
           form.knop2.colornormal = 'w/gb'
           form.knop2.enabled = .f.
           form.knop2.before.setfocus()
      CASE stuurelementId = 3
           form.knop3.colornormal='w/gb'
           form.knop3.enabled = .f.
           form.knop3.before.setfocus()
      CASE stuurelementId = 4
           form.knop4.colornormal='w/gb'
           form.knop4.enabled = .f.
           form.knop4.before.setfocus()
   ENDCASE
  RETURN

ENDCLASS
```

Gegevens valideren

Een belangrijke taak bij invoerformulieren is het valideren van invoer. Met de volgende actie en kenmerken kunt u de gegevens in invoervakken, ringvelden, keuzelijsten met invoervak en bladerobjecten valideren:

- *Valid*. De afhandelingsroutine voor deze actie wordt uitgevoerd wanneer u de waarde van een stuurelement wijzigt en het stuurelement vervolgens probeert te sluiten (bijvoorbeeld door te drukken op *Tab*). De afhandelingsroutine evalueert de nieuwe waarde en resulteert in .T. (waar) als de waarde geldig is of .F. (onwaar) als de waarde ongeldig is. Als het resultaat .F. is, kunt u het stuurelement niet sluiten.
- *ValidRequired*. Dit kenmerk bepaalt of de afhandelingsroutine voor de actie Valid wordt uitgevoerd telkens wanneer het stuurelement de focus verliest (zelfs wanneer u de standaardwaarde hebt laten staan bij het stuurelement) of alleen wanneer u wijzigingen aanbrengt.
- *ValidErrorMsg*. Dit kenmerk bepaalt de tekst van een melding die verschijnt op de statusbalk wanneer de afhandelingsroutine voor de actie Valid resulteert in .F. (onwaar).

De kenmerken Valid en ValidErrorMsg komen overeen met de opties VALID en ERROR van het commando @...SAY...GET in dBASE IV.

In het volgende voorbeeld ziet u hoe u gegevens valideert met deze actie en kenmerken. Als u een invoervak definieert waarin gebruikers een tijd kunnen opgeven, kunt u de volgende afhandelingsroutine voor de actie Valid van het invoervak opgeven om te zorgen dat gebruikers een geldige tijd invoeren.

```
DEFINE ENTRYFIELD Tijd OF Tijdform;              && Invoervak voor tijd
  PROPERTY;
    Left 15,;
    Width 5,;
    Picture "99:99",;
    Valid GldTijd,;                              && Actie-afhandelingsroutine toewijzen
    ValidRequired .T.,;                          && Geldige tijd vereisen
    ValidErrorMsg "Ongeldige tijd"               && Foutmelding toewijzen

PROCEDURE GldTijd                                && Procedure om tijd te valideren
   LOCAL Uur,Min                                 && LOCAL-variabelen voor tijdelijke
                                                 && waarden
   Uur=VAL(LEFT(this.Value,2))                   && Uren ophalen
   Min=VAL(RIGHT(this.Value,2))                  && Minuten ophalen
   IF Uur<0 .OR. Uur>23 .OR. Min<0 .OR. Min>59   && Testen op ongeldige waarden
      IsValid =.F.
   ELSE
      IsValid =.T.
   ENDIF
RETURN IsValid
```

Reageren op veranderingen

Wanneer u de waarde van een stuurelement wijzigt, naar een ander record gaat of een record toevoegt, kunt u het formulier hierop laten reageren door de acties OnChange, OnNavigate en OnAppend in te stellen.

OnChange

In het volgende voorbeeld wijst u de procedure KleurSaldo toe aan de actie OnChange van het invoervak Saldo. Als de waarde van het invoervak positief is, wordt de achtergrondkleur ingesteld op groen. Als de waarde negatief is, wordt de achtergrondkleur gewijzigd in rood.

```
DEFINE ENTRYFIELD Saldo OF AfschriftFormulier;
   PROPERTY;
      OnChange KleurSaldo,;                  && Actie-afhandelingsroutine toewijzen
      ColorNormal "",;
      ColorHighLight "",;
      Width         24.00,;
      Top          5.00,;
      Left          8.00,;
      Height          1.00,;
      Picture      "99999.99"

PROCEDURE KleurSaldo                         && Procedure om kleur van Saldo in te stellen
   IF Saldo.Value < 0                        && Als Saldo negatief is
      this.colorhighlight = "w+/r"           && Achtergrond instellen op rood
   ELSE                                      && Als Saldo positief is
      this.colorhighlight = "w+/g"           && Achtergrond instellen op groen
   ENDIF
RETURN
```

Op bladzijde 220 ziet u een voorbeeld van een procedure die u kunt toewijzen aan het kenmerk OnNavigate van dit invoervak Saldo.

Onchange, When

In het volgende voorbeeld maakt u een dialoogvenster om betalingen in te voeren. U selecteert de betaalwijze in een keuzelijst met invoervak. Op basis van de geselecteerde betaalwijze wordt het label van een invoervak gewijzigd van Betaling in Chequenummer of Creditcard-nummer door de afhandelingsroutine voor de actie OnChange van de keuzelijst met invoervak.

```
* Betaling.prg
DECLARE Betaalwijze[3]                    && Array om betaalwijzen weer te geven
Betaalwijze[1]="Contant"
Betaalwijze[2]="Cheque"
Betaalwijze[3]="Credit card"
LOCAL f
f = NEW Betaling()                         && Formulier maken
f.ReadModal()                              && Formulier openen
CLASS Betaling OF FORM
   this.Text = "Betaling invoeren"
   this.MDI = .F.
   this.Width =         40.00
   this.Top =          2.00
   this.Left =          2.00
   this.Height =         15.00
    DEFINE TEXT Tekst1 OF THIS;
       PROPERTY;
          ColorNormal "N/W",;
          Text         "Betaalwijze",;
          Width          len(this.text),;
          Top          2.00,;
          Left          4.00,;
          Height       2.00
    DEFINE COMBOBOX Keuzevak1 OF THIS;
       PROPERTY;
          Width          20.00,;
          Top          3,;
          Left         4.00,;
          Height          6.50,;
          Style          1,;
          DataSource "ARRAY Betaalwijze",;  && Keuzelijst met invoervak koppelen aan array
          Value Betaalwijze[1],;            && Waarde initialiseren met eerste element
          OnChange class::KeuzeWijzigen     && Actie-afhandelingsroutine toewijzen
    DEFINE TEXT Tekst2 OF THIS;
       PROPERTY;
          ColorNormal "N/W",;
          Text         " ",;
          Width          len(this.text),;
          Top          2.00,;
          Left         20.00,;
          Height       2.00
```

```
   DEFINE ENTRYFIELD Invoervak1 OF THIS;
      PROPERTY;
         Width         17.00,;
         Top          3.00,;
         Left       20.00,;
         Height          2.00,;
         Value         0,;
         Border .T.,;
         When {form.Tekst2.Text <> " "}    && Alleen beschikbaar als betaalwijze niet
                                           && contant is
   DEFINE TEXT Tekst3 OF THIS;
      PROPERTY;
         ColorNormal "N/W",;
         Text         "Betaald bedrag",;
         Width          len(this.text),;
         Top          6.00,;
         Left         20.00,;
         Height       2.00
   DEFINE ENTRYFIELD Invoervak2 OF THIS;
      PROPERTY;
         Width         17.00,;
         Top          7.00,;
         Left       20.00,;
         Height          2.00,;
         Value         0,;
         Border .T.,;
         Picture "999999.99"

PROCEDURE KeuzeWijzigen                    && Actie-afhandelingsroutine voor keuzelijst
                                           && met invoervak
  DO CASE                                  && Label instellen voor Invoervak1
     CASE this.Value = "Contant"
        form.Tekst2.Text = " "
     CASE this.Value = "Cheque"
        form.Tekst2.Text = "Chequenummer"
     CASE this.Value = "Credit card"
        form.Tekst2.Text = "Creditcard-nummer"
  ENDCASE
RETURN
ENDCLASS
```

OnNavigate

In het volgende voorbeeld wijst u een codeblok toe aan de actie OnNavigate van een bladerobject. Het kenmerk ShowRecNo is ingesteld op .F. (onwaar) om de kolom met recordnummers weg te laten. In plaats daarvan wordt het kenmerk Text van een tekstobject ingesteld door de afhandelingsroutine voor de actie OnNavigate, om het huidige recordnummer weer te geven.

```
DEFINE BROWSE Blader1 OF THIS;
   PROPERTY;
      OnNavigate {;form.RecWeerg.Text = "Record:"+STR(RECNO(),6)},;
      Width         49.25,;
      Top         15.00,;
```

```
Left          7.00,;
Height          9.13,;
ShowRecNo .F.,;
Alias "KLANTEN"
```

In het volgende voorbeeld wijzigt u de kleur van het invoervak Openstaand_bedrag dat is gekoppeld aan een debiteurentabel. Als het bedrag positief is, wordt de achtergrondkleur van het invoervak ingesteld op groen. Als het bedrag negatief is, wordt de achtergrondkleur gewijzigd in rood. U kunt deze actie-afhandelingsroutine bijvoorbeeld toewijzen aan de actie OnNavigate van een formulier met het invoervak Openstaand Bedrag.

```
PROCEDURE KleurSaldo
   IF form.Openstaand_bedrag.Value < 0          && Als Saldo negatief is
      this.colorhighlight = "w+/r"              && Achtergrond instellen op rood
   ELSE                                         && Als Saldo positief is
      this.colorhighlight = "w+/g"              && Achtergrond instellen op groen
   ENDIF
RETURN
```

Focusverplaatsing afhandelen

In de meeste formulieren kunt u een focusverplaatsing afhandelen met de kenmerken StatusMessage of ColorHighlight om een melding weer te geven of de kleur te wijzigen. Als u dit niet voldoende vindt, kunt u acties voor fcosuverplaatsing instellen voor de meeste objecten en het formuliervenster zelf met de kenmerken OnGotFocus en OnLostFocus.

U kunt bijvoorbeeld de beschikbare menu-opties wijzigen op basis van het geselecteerde stuurelement. In het volgende voorbeeld ziet u actie-afhandelingsroutines voor de kenmerken OnGotFocus en OnLostFocus van een bladerobject. Wanneer het bladerobject de focus krijgt, wordt het vooraf gedefinieerde menu Bewerken beschikbaar. Zodra het bladerobject de focus verliest, wordt het menu lichter gekleurd weergegeven en is dit niet langer beschikbaar.

```
PROCEDURE BladerMenus                  && Toewijzen aan OnGotFocus van bladerobject
   form.Hoofdmenu.Bewerken.Enabled = .T.
RETURN

PROCEDURE GeenBladerMenus              && Toewijzen aan OnLostFocus van bladerobject
   form.Hoofdmenu.Bewerken.Enabled = .F.
RETURN
```

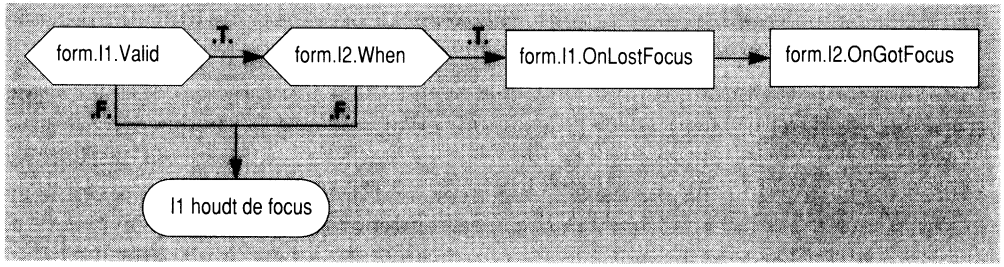
Wanneer formulieren of stuurelementen de focus verliezen, kunnen veel acties plaatsvinden. In de volgende twee secties wordt de actievolgorde besproken.

Actievolgorde bij verplaatsing tussen stuurelementen

U start acties wanneer u de focus verplaatst van het ene stuurelement naar een ander stuurelement in een formulier.

In Afbeelding 14.11 ziet u een overzicht van de actievolgorde wanneer een invoervak (I1) de focus heeft en u op een ander invoervak (I2) klikt.

Afbeelding 14.11 Verplaatsen tussen stuurelementen in hetzelfde forrmulier



Wanneer invoervak I1 de focus heeft en u drukt op *Tab* om naar I2 te gaan, gebeurt het volgende:

1 De afhandelingsroutine voor de actie Valid van I1 wordt uitgevoerd. Als de routine resulteert in .T. (waar), wordt stap 2 uitgevoerd. Als de routine resulteert in .F. (onwaar), is de invoer in I1 ongeldig en houdt I1 de focus.

2 De afhandelingsroutine voor de actie When van I2 wordt uitgevoerd. Als de routine resulteert in .T. (waar), wordt stap 3 uitgevoerd. Als de routine resulteert in .F. (onwaar), kan I2 niet de focus krijgen en houdt I1 de focusE2.

3 I1 heeft zojuist de focus verloren, dus wordt de afhandelingsroutine voor de actie OnLostFocus van I1 uitgevoerd. I2 krijgt nu de focus.

4 I2 heeft zojuist de focus gekregen, dus wordt de afhandelingsroutine voor de actie OnGotFocus van I2 uitgevoerd.

Actievolgorde bij verplaatsing tussen formulieren

Wanneer u formulieren opent met het commando OPEN FORM, kunt u twee of meer formulieren tegelijkertijd weergeven. Hoewel alle open formulieren beschikbaar zijn, heeft slechts één formulier de focus. U kunt een formulier de focus geven door te klikken in het formulier of te drukken op *Ctrl+F6* om te schakelen tussen open MDI-formulieren.

Wanneer u de focus verplaatst van het ene formulier (het huidige formulier) naar een ander formulier (het nieuwe formulier), gebeurt het volgende:

1 Het huidige formulier verliest de focus en de afhandelingsroutine voor de actie OnLostFocus van het huidige formulier wordt uitgevoerd.

2 Het nieuwe formulier krijgt de focus en de afhandelingsroutine voor de actie OnGotFocus van het nieuwe formulier wordt uitgevoerd.

3 Wanneer een formulier de focus krijgt, heeft altijd een van de stuurelementen de focus:

- Als u de focus hebt verplaatst zonder een stuurelement in het nieuwe formulier te selecteren, door bijvoorbeeld te drukken op *Ctrl+F6*, wordt de focus in het nieuwe formulier niet verplaatst en vindt er geen verdere uitvoering plaats.
- Als u een stuurelement in het nieuwe formulier selecteert, door bijvoorbeeld te klikken op een stuurelement, wordt de actievolgorde uitgevoerd die is beschreven in "Actievolgorde bij verplaatsing tussen stuurelementen".

Help instellen

De meeste applicaties beschikken over een vorm van Help. De eenvoudigste vorm is een tekstobject dat de inhoud van een ander object aangeeft, zoals "Naam" naast een invoervak voor namen. U kunt ook meldingen weergeven op de statusbalk met het kenmerk StatusMessage, die verschijnen wanneer het object de focus krijgt.

Als u uitgebreidere Help wilt instellen, kunt u Help-vensters weergeven met de actie OnHelp. Deze actie wordt uitgevoerd wanneer u drukt op *F1*. Wijs een afhandelingsroutine toe aan de actie Onhelp van het formulier om Help weer te geven voor het gehele formulier, of wijs een afhandelingsroutine toe aan de actie OnHelp van elk stuurelement om contextgebonden Help weer te geven.

U kunt bijvoorbeeld een tabel met Help-teksten maken, waarbij u één record maakt voor elk formulier in de toepassing en de Help-tekst opslaat in een memoveld. Vervolgens wijst u de volgende procedure toe aan de actie OnHelp van elk formulier:

```
PROCEDURE HelpWeergeven
   HuidigWerkgebied = WORKAREA()      && Huidig werkgebied opslaan
   SELECT HelpTab                     && HELPTAB.DBF openen met alias HelpTab
   SEEK(this.Text)                    && Formulierkenmerk Text instellen als zoeksleutel
   IF FOUND()                         && Formulier HelpVenster weergeven (met
      OPEN FORM Helpvenster           && editor-object dat is gekoppeld aan memoveld)
   ENDIF
   SELECT (HuidigWerkgebied)          && Teruggaan naar oorspronkelijk werkgebied
RETURN
```

Programmeurs die ervaring hebben met de Help-functie van Windows, kunnen een Windows-Helpbestand (.HLP) maken voor een dBASE-applicatie. U hebt hiervoor Windows Help Compiler nodig. In een .HLP file wordt elk Help-scherm ("onderwerp") geïdentificeerd met een *Help-sleutelwoord*. U kunt een .HLP-bestand opgeven voor een formulier met de kenmerken HelpFile en HelpId:

- *HelpFile*. Bij dit kenmerk geeft u de naam op van het .HLP-bestand met de Help-onderwerpen voor het object.
- *HelpId*. Bij dit kenmerk geeft u het Help-sleutelwoord op van het onderwerp met Help voor het object.

Wanneer u de kenmerken HelpFile en HelpId hebt ingesteld, kunt u op *F1* drukken om de Windows Help-toepassing (WINHELP.EXE) te starten en het opgegeven onderwerp in het .HLP-bestand weer te geven.

Acties voor formulierverplaatsing afhandelen

U kunt formuliervensters verplaatsen en het formaat ervan wijzigen, tenzij u de formulierkenmerken Moveable en Sizeable instelt op .F. (onwaar). Bij de meeste formulieren hoeft u geen actie-afhandelingsroutines in te stellen om het formulier te laten reageren op deze acties. U kunt echter ook formulieren maken die op een bepaalde manier moeten worden geplaatst door code toe te wijzen aan de acties OnMove en OnSize.

OnMove, OnSize

In het volgende voorbeeld maakt u twee formulieren, Asterix en Obelix. U kunt Obelix niet verplaatsen, maar als u Asterix verplaatst, volgt Obelix automatisch. De procedure AsterixVerpl wordt toegewezen aan de actie OnMove en OnSize van Asterix.

```
* ASTERIX.PRG
SET PROCEDURE TO PROGRAM(1) ADDITIVE
Asterix = NEW FORM()
Asterix.Text = "Asterix"
Asterix.Width  =        25
Asterix.Top    =       2.00
Asterix.Left   =        2.00
Asterix.Height =         15.00
Asterix.MDI = .F.
Asterix.OnMove = AsterixVerpl
Asterix.OnSize = AsterixVerpl
Asterix.OnClose = {;this.Obelix.Close()}

Asterix.Obelix = NEW FORM()
Asterix.Obelix.Top = Asterix.Top
Asterix.Obelix.Width=  20
Asterix.Obelix.left = Asterix.Left+Asterix.Width+1
Asterix.Obelix.height=   5
Asterix.Obelix.MDI= .F.
Asterix.Obelix.moveable = .F.
Asterix.Obelix.Showname = NEW TEXT(Asterix.Obelix)
Asterix.Obelix.Showname.Text = "Obelix"
Asterix.Open()
Asterix.Obelix.Open()

PROCEDURE AsterixVerpl(nLeft,nTop)
   this.Obelix.left=this.left+this.width +1
   this.Obelix.top=this.top
RETURN
```

OnSize

Dit voorbeeld is afkomstig uit DBKLOK.PRG in de directory VOORBD, waarmee de tijd wordt weergegeven in een venster. Wanneer u het formaat van het klokvenster wijzigt, wordt het formaat van de tekst in het venster (de tijd) aangepast door de procedure ReSizeText.

```
PROCEDURE ReSizeText
   local t                                    && Nieuwe verwijzing naar tekstobject maken,
   t = form.timeText                          && om korter verwijzingspad te definiëren.
   t.width = form.width                       && Breedte aanpassen aan formulierbreedte
   t.height = form.height                     && Hoogte aanpassen aan formulierhoogte
   t.fontsize=(form.width+form.height)/2*1.5  && Fontgrootte aanpassen
RETURN
```

Muisacties afhandelen

In de meeste formulieren kunt u klikacties afhandelen met OnClick. Deze actie is beschikbaar voor knoppen en menu-opties. Als u dit niet voldoende vindt, kunt u de acties in Tabel 14.1 gebruiken om klikhandelingen af te handelen voor elk formulierobject, bijvoorbeeld dubbelklikken, drukken op muisknoppen en muisknoppen loslaten. Bovendien kunt u klikacties afhandelen die worden gecombineerd met bepaalde toetsen, zoals *Ctrl* of *Alt*.

Tabel 14.1 Muisacties

Muisknop	Dubbelklikken	Drukken op	Loslaten
Links	OnLeftDblClick	OnLeftMouseDown	OnLeftMouseUp
Midden	OnMiddleDblClick	OnMiddleMouseDown	OnMiddleMouseUp
Rechts	OnRightDblClick	OnRightMouseDown	OnRightMouseUp

De volgende drie parameters worden overgebracht naar de acties in Tabel 14.1 (en de acties OnSize en Key) :

- *Vlaggen*. Deze parameter is één-byte-waarde die aangeeft of een gebruiker heeft gedrukt op *Ctrl, Alt* of een van de andere muisknoppen, terwijl de actie plaatsvond. Gebruik de bitbewerkingsfuncties, zoals BITSET() of BITAND() om de waarde te interpreteren.
- *Kolom*. Deze parameter geeft de kolompositie van de muisaanwijzer aan.
- *Rij*. Deze parameter geeft de rijpositie van de muisaanwijzer aan.

Gebruik de normale syntaxis voor ontvangst van parameters in de code voor actie-afhandelingsroutines (zie Hoofdstuk 4). Zie *Commando's en functies* voor meer informatie over het gebruik van de parameter Vlaggen bij de muisacties.

In het volgende voorbeeld gebruikt u de functie BITAND() om de parameter Vlag te interpreteren en de toets weer te geven waarop een gebruiker drukt. U kunt deze procedure koppelen aan elke actie in Tabel 14.1.

```
PROCEDURE MuisvlaggenTonen(nVlag,nKol,nRij)
   nShift=4
   nCtrl =8
   IF BITAND(nVlag,nShift)= nShift
      ? " Shift-toets"
   ENDIF
   IF BITAND(nVlag,nCtrlKey)=nCtrl
      ? " Ctrl-toets"
   ENDIF
RETURN
```

Hoofdstuk

15

Eigen stuurelementen maken

Een *eigen stuurelement* is een object voor de gebruikersinterface, net als invoervakken en knoppen, met een zelf-gedefinieerd uiterlijk of werking. U kunt eigen dBASE-stuurelementen maken door een klasse te definiëren op basis van een ingebouwde klasse. U kunt ook VBX-stuurelementen gebruiken als eigen stuurelementen. In dit hoofdstuk wordt beschreven wat eigen stuurelementen zijn en hoe u eigen stuurelementen gebruikt in formulieren, en worden er voorbeelden gegeven van stuurelementen die u kunt maken.

Werken met eigen stuurelementen

Met eigen stuurelementen kunt u het samenstellen van formulieren vereenvoudigen door objecten te maken die geschikt zijn om opnieuw te gebruiken, of actie-afhandelingsroutines te maken die vooraf zijn ingesteld op bepaalde waarden. U kunt bijvoorbeeld stijlconventies ontwikkelen voor uw formulieren en altijd hetzelfde font en dezelfde kleurkenmerken gebruiken voor invoervakken. Als u een eigen stuurelement voor invoervakken maakt waarbij u deze kenmerken vooraf instelt, hoeft u deze kenmerken niet apart in te voeren voor elk invoervak. Vervolgens plaatst u uw eigen objecten in uw formulieren in plaats van de standaardobjecten.

Eigen dBASE-stuurelementen zijn klassedefinities met eigen instellingen voor kenmerken en actie-afhandelingsroutines. U kunt de klassecode zelf schrijven, of de code aanpassen die wordt gegenereerd door Formulierontwerp. Het bestand KNOPPEN.CC in de directory VOORBD bevat klassen voor eigen stuurelementen voor algemene knoppen, zoals **OK** en **Annuleren**.

VBX-stuurelementen zijn objecten voor de gebruikersinterface die zijn geschreven in C of Pascal en vervolgens zijn gecompileerd als speciaal DLL-bestand met de extensie .VBX. Net als eigen dBASE-stuurelementen zijn VBX-stuurelementen objecten die het uiterlijk van algemene stuurelementen (zoals keuzelijsten en keuzerondjes) verfraaien of de werking ervan uitbreiden. Een aankruisvakje geeft bijvoorbeeld aan of een bepaalde eigenschap is ingeschakeld of uitgeschakeld. Het VBX-stuurelement

SCHAKEL.VBX heeft dezelfde functie, maar een fraaier uiterlijk (het object lijkt op een lichtschakelaar). Veel externe leveranciers brengen bibliotheken met VBX-stuurelementen op de markt.

dBASE voor Windows ondersteunt VBX-stuurelementen versie 1.

Eigen stuurelementen installeren

U kunt uw eigen stuurelementen installeren, zodat de stuurelementen verschijnen op het tabblad **Eigen** van het stuurelementenpalet in Formulierontwerp. U kunt de eigen stuurelementen vervolgens in formulieren plaatsen, op dezelfde manier als standaardstuurelementen.

U installeert als volgt eigen dBASE-stuurelementen via het menu:

1 Sla de eigen dBASE-klassen op in een programmabestand met de extensie .CC (CC verwijst naar de Engelse term voor eigen stuurelement, "custom control").

2 Open Formulierontwerp en kies **Bestand | Eigen stuurelementen instellen.**

3 Selecteer het .CC-bestand in het dialoogvenster dat verschijnt.

Wanneer u een eigen stuurelement installeert, wordt er automatisch een regel toegevoegd aan de sectie [CustomClasses] van het configuratiebestand, DBASEWIN.INI. U kunt deze regel ook zelf toevoegen aan DBASEWIN.INI, in plaats van de stuurelementen te installeren via het menu.

In het volgende voorbeeld ziet u een opdracht waarmee een .CC-bestand wordt geïnstalleerd in DBASEWIN.INI:

```
[CustomClasses]                          && Begin van sectie CustomClasses
CC0=C:\DBASEWIN\VOORBD\KNOPPEN.CC        && KNOPPEN.CC installeren
CC1=C:\EIGOBJ\BIBL1.CC                   && BIBL1.CC installeren
```

De volgende syntaxis is van toepassing op regels in DBASEWIN.INI waarmee bestanden met eigen stuurelementen worden geïnstalleerd:

```
CCn  = <.CC bestandsnaam>
```

Hierbij is *n* een getal tussen 0 en 99. Het nummer *n* heeft geen speciale betekenis. Dit is een intern nummer dat wordt gebruikt door dBASE. (Zie Appendix C in het *Handboek* voor meer informatie over de instellingen in DBASEWIN.INI.)

Als u VBX-stuurelementen wilt installeren, volgt u dezelfde stappen als voor eigen dBASE-stuurelementen en selecteert u het .VBX-bestand dat u wilt installeren. Omdat .VBX-bestanden een speciaal type DLL-bestand zijn, wordt er een regel toegevoegd aan de sectie [DLLS] van DBASEWIN.INI, in plaats van de sectie [CustomClasses] (zie volgend voorbeeld).

```
[DLLs]                                   && Begin van sectie DLLs
DLL0=C:\DBASEWIN\VOORBD\DBTIMER.VBX      && Stuurelement DBTIMER installeren
DLL1=C:\DBASEWIN\VOORBD\SCHAKEL.VBX      && Stuurelement SCHAKEL installeren
```

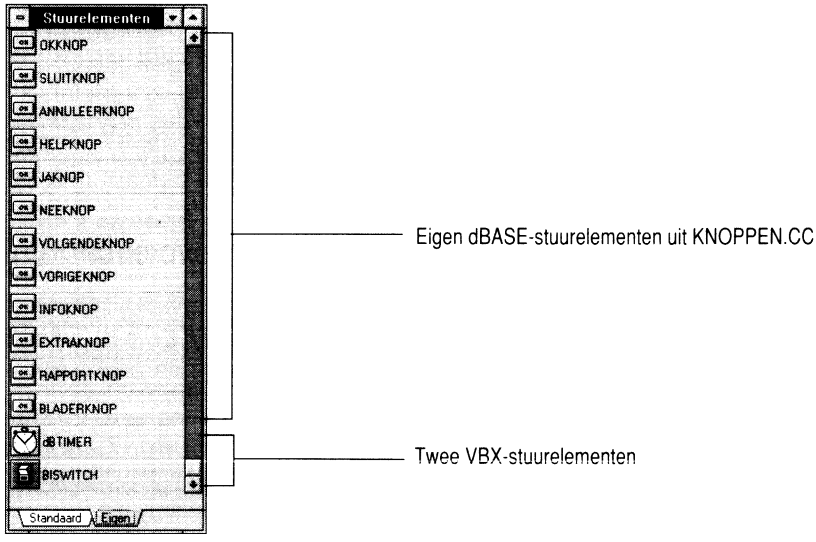
U kunt VBX-stuurelementen ook individueel installeren met het commando LOAD DLL In het volgende voorbeeld ziet u de code waarmee dezelfde VBX-stuurelementen

worden geladen als in het vorige voorbeeld. De stuurelementen worden nu echter geladen in een programmabestand (of het commandovenster) en niet in DBASEWIN.INI.

```
LOAD DLL C:\DBASEWIN\VOORBD\DBTIMER.VBX        && Stuurelement DBTIMER installeren
LOAD DLL C:\DBASEWIN\VOORBD\SCHAKEL.VBX        && Stuurelement SCHAKEL installeren
```

In Afbeelding 15.1 ziet u hoe de eigen dBASE-stuurelementen uit KNOPPEN.CC en de VBX-stuurelementen SCHAKEL.VBX en DBTIMER.VBX worden weergegeven in het stuurelementenpalet, nadat deze stuurelementen zijn geïnstalleerd.

Afbeelding 15.1 Het tabblad Eigen van het stuurelementenpalet



Klassen definiëren voor eigen stuurelementen

In Hoofdstuk 11 wordt uitgebreid behandeld hoe u zelf klassedefinities schrijft. U kunt de klassedefinitie voor een stuurelement ook op een andere manier genereren die veel gemakkelijker is: u maakt het stuurelement eerst op met Formulierontwerp en past vervolgens de code bij de opdracht DEFINE in het .WFM-bestand aan om een klassedefinitie te maken voor een .CC-bestand.

In Afbeelding 15.2 ziet u een vergelijking tussen de opdracht DEFINE voor een invoervak in een .WFM-bestand, en een identieke klassedefinitie in een .CC-bestand.

Afbeelding 15.2 Vergelijking tussen DEFINE-code in .WFM-bestand en CLASS-code in .CC-bestand

DEFINE-code uit .WFM-bestand

```
DEFINE ENTRYFIELD Eigen OF THIS;
   PROPERTY;
      ColorNormal       "N/GB",;
      MousePointer            3,;
      Value                  "",;
      FontBold               .F.,;
      FontSize             11.00,;
      FontName     "Helvetica",;
      Border                 .T.,;
      ColorHighLight     "N/GB+"
```

CLASS-code uit .CC-bestand

```
CLASS Eigen(f,n) OF ENTRYFIELD(f,n) CUSTOM
      this.ColorNormal    = "N/GB"
      this.MousePointer   =  3
      this.Value          = ""
      this.FontBold       = .F.
      this.FontSize       = 11.00
      this.FontName       = "Helvetica"
      this.Border         = .T.
      this.ColorHighLight = "N/GB+"
ENDCLASS
```

U converteert als volgt de opdracht DEFINE naar een klassedefinitie voor een eigen stuurelement:

1 Wijzig de opdracht `DEFINE` in de opdracht `CLASS` (zie in Afbeelding 15.2).
   - Het sleutelwoord `CUSTOM` aan het einde van de regel geeft aan dat er een eigen stuurelement wordt gemaakt met deze klassedefinitie.

Belangrijk
   - De parameters *f* en *n* ontvangen een verwijzing naar het formulier en de naam van het stuurelement. Normaal gesproken hoeft u alleen de formulierparameter op te geven om een subklasse van een standaardklasse te definiëren. Klassen voor eigen stuurelementen vereisen echter een extra parameter voor de naam van het stuurelement. U kunt een willekeurige naam toewijzen aan deze extra parameter, net als aan de formulierparameter. In dit voorbeeld wordt *n* gebruikt.

2 Verwijder de regel `PROPERTY;`

3 Voer de volgende handelingen uit voor elke regel waarin een kenmerk wordt toegewezen:
   - Voeg `this.` toe vóór de naam van het kenmerk.
   - Voeg een gelijkteken (=) toe na de naam van het kenmerk.
   - Verwijder `,;` aan het einde van de regel.

4 Als u procedures hebt gemaakt voor actie-afhandelingsroutines, voegt u de proceduredefinitie toe na de laatste kenmerktoewijzing.

5 Voeg een regel `ENDCLASS` toe aan het einde om de klassedefinitie af te sluiten.

Voorbeelden van eigen dBASE-stuurelementen

Hierna ziet u een aantal voorbeelden van klassedefinities voor eigen stuurelementen:

Knop OK
Met de volgende klasse (uit KNOPPEN.CC in de directory VOORBD) definieert u de eigen knop **OK**.

```
CLASS OkKnop(f,n) of PUSHBUTTON(f,n) CUSTOM
   this.Height = 1.50
   this.Width = 15.00
   this.UpBitmap = "Resource #20"
   this.DisabledBitmap = "Resource #21"
   this.Text = "OK"
ENDCLASS
```

Invoervak Wachtwoord

In dit voorbeeld maakt u een invoervak voor een wachtwoord. U gebruikt het kenmerk Key om elke toetsaanslag op te sporen en in plaats daarvan een asterisk (*) weer te geven.

```
CLASS Wachtwoord(f,n) OF ENTRYFIELD(f,n) CUSTOM
   this.ColorNormal = ""
   this.Value = ""
   this.Border = .T.
   this.ColorHighLight = ""
   this.Width = 8
   this.Height = 2

   this.EchteWaarde = ""

   PROCEDURE Key(nTek, TellingHerhalen, vlaggen)
      this.EchteWaarde = this.EchteWaarde + chr(nTek)
      this.Value = this.Value + "*"
   RETURN .F.
ENDCLASS
```

Ringveld voor recordnavigatie

In dit voorbeeld maakt u een ringveld om de recordwijzer in een tabel te verplaatsen. U kunt dit ringveld in een formulier plaatsen dat is gekoppeld aan een tabel, om door de tabelrecords te bladeren. Dit ringveld komt vooral van pas als u het dBASE-menu en de knoppenbalken hebt uitgeschakeld met SHELL().

```
CLASS RecordRing(f,n) OF SPINBOX(f,n) CUSTOM
   this.Value    = RECNO()              && Kenmerk Value instellen op recordnummer
   this.RangeMin = 1                    && Laagste recordnummer instellen op 1
   this.RangeMax = RECCOUNT()           && Hoogste recordnummer instellen
   PROCEDURE OnChange
      GOTO this.Value                   && Recordaanwijzer verplaatsen naar Value
   RETURN
ENDCLASS
```

VBX-stuurelementen gebruiken

U kunt VBX-stuurelementen plaatsen in formulieren en de bijbehorende kenmerken, acties en methoden instellen op dezelfde manier als standaardstuurelementen. De leverancier van VBX-stuurelementen is verantwoordelijk voor Help bij specifieke kenmerken en het gebruik van de VBX-stuurelementen. dBASE biedt uitsluitend middelen om de stuurelementen te plaatsen in dBASE-formulieren.

De beschikbare kenmerken, acties en methoden zijn gedefinieerd in het .VBX-bestand. Bepaalde kenmerken, zoals Top en Left, kunnen dezelfde namen hebben als de corresponderende kenmerken voor dBASE-stuurelementen.

Daarnaast worden de volgende standaardkenmerken toegevoegd aan elk VBX-stuurelement (zie *Commando's en functies* voor meer informatie over deze kenmerken):

- *hWnd*. Met dit kenmerk wordt een handle ingesteld naar het formuliervenster waarin u het stuurelement plaatst. De handle is een unieke numerieke waarde die wordt toegewezen door Windows.
- *Parent*. Met dit kenmerk wordt een verwijzing ingesteld naar het formuliervenster waarin u het stuurelement plaatst.
- *Before*. Met dit kenmerk wordt een verwijzing ingesteld naar het object dat voorafgaat aan het stuurelement in de tabvolgorde.

Bovendien worden de volgende VBX-kenmerken toegevoegd:

- *VBStream*. Met dit kenmerk wordt informatie opgeslagen die wordt gebruikt door het VBX-stuurelement om het .BMF-bestand van Formulierontwerp te lezen. Als u namelijk een formulier met een VBX-stuurelement opslaat, wordt er een extra bestand gemaakt (*<formuliernaam>*.BFM) met binaire informatie over de beginstand van alle VBX-stuurelementen in het formulier.

Belangrijk Het .BFM-bestand moet beschikbaar zijn wanneer u het formulier start

- *FormGetsKeys*. Met dit logische kenmerk wordt ingesteld of de volgende navigatietoetsen worden toegewezen aan het formulier of aan het VBX-stuurelement: *Ctrl+End, Ctrl+W, Enter, Esc, Tab, Shift+Tab*. De standaardwaarde is .T. (waar), wat aangeeft dat de toetsaanslagen worden toegewezen aan het formulier.

  U kunt het kenmerk FormGetsKeys bijvoorbeeld instellen op .F. (onwaar) voor VBX-stuurelementen die gebruik maken van deze toetsen, bijvoorbeeld stuurelementen voor tekstverwerkingsprogramma's of spreadsheet's. Als FormGetsKeys .F. is, worden de toetsen toegewezen aan het VBX-stuurelement en niet aan het formulier

- *Translate*. Met dit logische kenmerk wordt ingesteld of OEM-tekenreeksen moeten worden omgezet naar ANSI en omgekeerd. De standaardwaarde is .T. (waar).

  dBASE voor Windows gebruikt de OEM-tekenset om compatibel te blijven met eerdere dBASE-versies. VBX-stuurelementen gebruiken echter de ANSI-tekenset. Wanneer Translate is ingesteld op .T., worden de tekenreeksen die worden toegewezen aan de kenmerken van het VBX-stuurelement, omgezet van OEM naar ANSI. De tekenreeksen die worden opgehaald uit het stuurelement, worden omgezet van ANSI naar OEM.

  Meestal hoeft u het kenmerk Translate niet in te stellen. Stel Translate alleen in op .F. (onwaar) als het stuurelement binaire gegevens of tekenwaarden bevat die intact moeten blijven om bepaalde redenen.

In het volgende voorbeeld maakt u een klassedefinitie voor een formulier met het VBX-stuurelement SCHAKEL.VBX (een schakelaar). Wanneer de schakelaar is ingeschakeld, wordt het formulier rood. Het formulier wordt wit wanneer de schakelaar is uitgeschakeld.

```
CLASS SCHAKELFORM OF FORM
   this.HelpId = ""
   this.HelpFile = ""
   this.Text = "Schakelaar"
   this.Height =          6.78
   this.Left =          24.90
   this.Top =           5.23
   this.Width =          25.73
   load dll SCHAKEL.VBX
   DEFINE BISWITCH BISWITCH1 OF THIS;
       PROPERTY;
         OnOpen CLASS::BISWITCH1_ONOFF,;
         Height          3.03,;
         Left          8.30,;
         Top           2.02,;
         VBStream "SCHAKEL.BFM 56",;
         Width          9.96,;
         OnOn CLASS::BISWITCH1_ONON,;
         OnOff CLASS::BISWITCH1_ONOFF
   Procedure BISWITCH1_OnOn
   this.caption = "AAN"
   form.colorNormal = "N/R+"
   Procedure BISWITCH1_OnOff
   this.caption = "UIT"
   form.colorNormal = "N/W"
ENDCLASS
```

In Afbeelding 15.3 ziet u het formulier SCHAKEL.

Afbeelding 15.3 Het formulier SCHAKEL

Als u klikt op de schakelaar, wordt de achtergrondkleur van het formuliervenster gewijzigd en verandert de tekst naast de schakelaar

Hoofdstuk 16

Werken met gegenereerde code

Formulierontwerp en Menu-ontwerp genereren dBASE-code op basis van de formulieren en menu's die u opmaakt. Voor de meeste projecten hoeft u de gegenereerde code niet te bewerken. U kunt de gegenereerde code echter aanpassen en uitbreiden als uw applicatie dit vereist.

Zowel Formulierontwerp als Menu-ontwerp zijn tweeweghulpmiddelen. U kunt de gegenereerde code aanpassen met een tekst-editor en vervolgens opnieuw laden in het ontwerpvenster. U kunt programma's dus op uw eigen manier ontwikkelen door objecten op te maken, de code aan te passen en terug te gaan naar de ontwerpvensters zo vaak als u wilt.

In dit hoofdstuk worden onderwerpen behandeld voor programmeurs die willen werken met de code die wordt gegenereerd door Formulierontwerp en Menu-ontwerp.

Werken met formulierontwerpcode

Formulierontwerp genereert een .WFM-bestand, dat bestaat uit de volgende drie hoofdsecties:

- *Programma-aanhef.* Dit is de eerste sectie. In de procedure-editor geeft u de aanhef op door **Aanhef** te selecteren in de keuzelijst met invoervak (zie Afbeelding 16.1). In de aanhef kunt u willekeurige code opgeven die u wilt uitvoeren voordat het formulier wordt gemaakt. Meestal bevat de aanhef de volgende onderdelen:
  - Opmerkingen. Hiermee beschrijft u het formulier. Zie Hoofdstuk 3 voor meer informatie over het invoeren van opmerkingen in programma's.
  - De opdracht CREATE SESSION. Met deze opdracht start u een formuliersessie met een nieuwe verzameling werkgebieden. U kunt CREATE SESSION gebruiken voor formulieren die zijn gekoppeld aan een tabel, om te zorgen dat het formulier niet wordt beïnvloed als de tabel wordt gewijzigd in Navigator of door een ander programma dat gelijktijdig wordt uitgevoerd (waarmee bijvoorbeeld de tabel

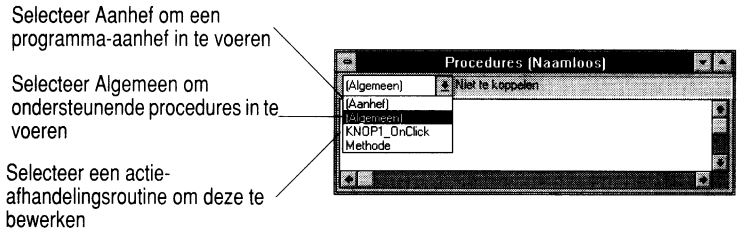
wordt gesloten of een filter wordt ingesteld). Zie Hoofdstuk 21 voor meer informatie over sessies.

- Preprocessor-instructies. Met deze instructies definieert u constanten of voegt u include-bestanden in. Zie Hoofdstuk 7 voor meer informatie over preprocessor-instructies.
- Definities voor geheugenvariabelen. Zie Hoofdstuk 5 voor meer informatie over geheugenvariabelen.

- *Klassedefinities*. De klassedefinities voor het formulier vormen de kern van de code. Formulierontwerp genereert de meeste subklassen voor u, zoals de kenmerkinstellingen van het formuliervenster en de definities van elk stuurelement. U voert actie-afhandelingsprocedures in met de procedure-editor.
- Definities voor *ondersteunende procedures*. Deze definities verschijnen na de klassedefinitie. Ondersteunende procedures zijn procedures die worden aangeroepen door actie-afhandelingsprocedures. In de procedure-editor geeft u ondersteunende procedures op door **Algemeen** te selecteren in de keuzelijst met invoervak (zie Afbeelding 16.1).

In Afbeelding 16.1 ziet u waar u de code voor elke sectie invoert in de procedure-editor.

Afbeelding 16.1 De procedure-editor



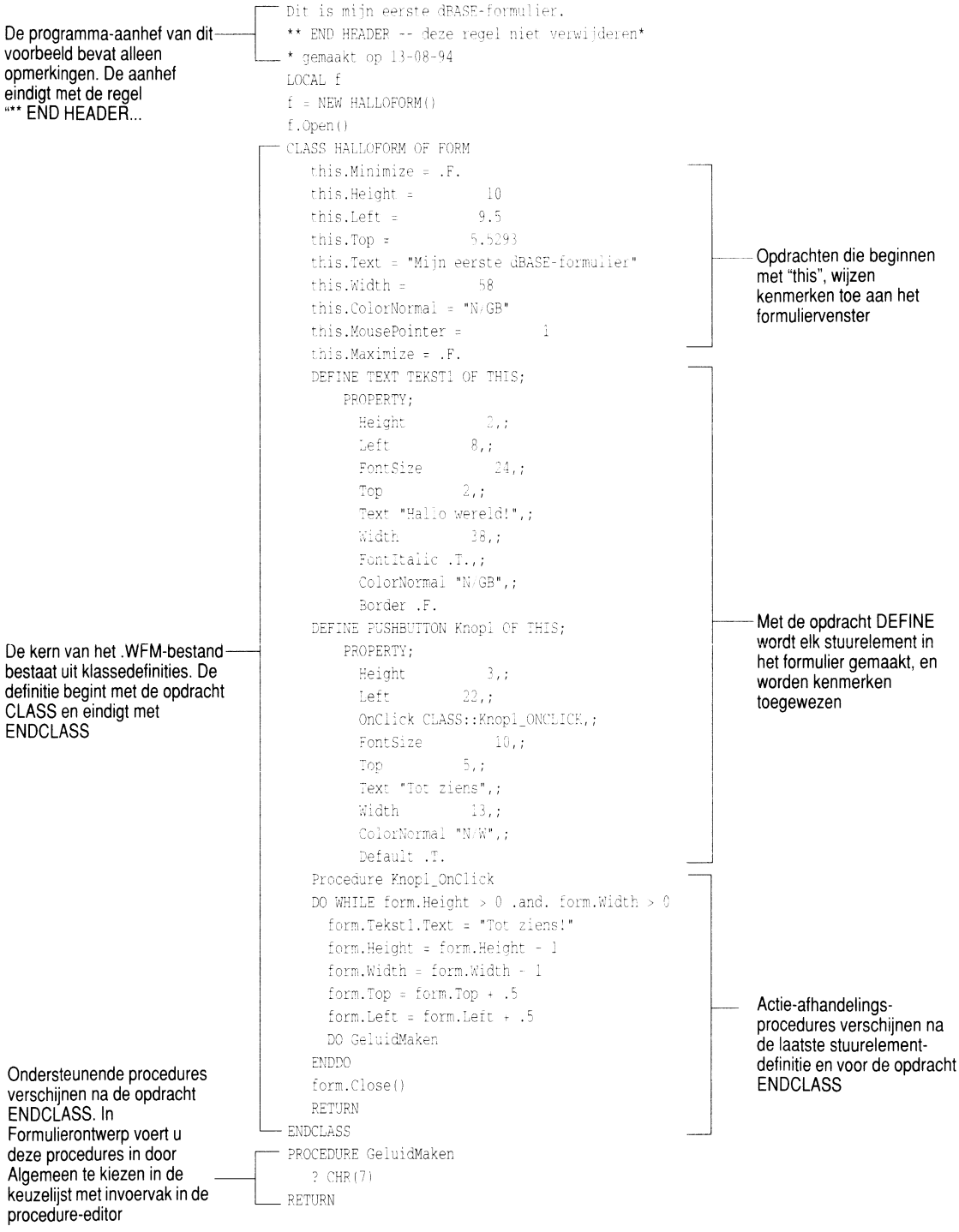
Formulierontwerp is een veelzijdig hulpmiddel om programmacode te genereren en wijzigen. Met Formulierontwerp kunt u een .WFM-bestand genereren, dit bestand vervolgens bewerken met een tekst-editor en teruggaan naar Formulierontwerp om het formulier visueel te bewerken.

U kunt ook uw eigen code schrijven en deze code visueel bewerken met Formulierontwerp. De oorspronkelijke code waarmee het formulier is gemaakt (commando's in het commandovenster of een programmabestand), wordt genegeerd. Wanneer u het formulier opslaat, wordt een nieuw .WFM-bestand gemaakt op basis van de huidige instellingen voor het formulier.

Let op de volgende punten als u code wijzigt met een tekst- editor en vervolgens Formulierontwerp start:

- Formulierontwerp genereert altijd een klassedefinitie voor het formulier, ongeacht hoe het formulier is gemaakt met de oorspronkelijke code.

Afbeelding 16.2 Uitleg bij een WFM-bestand van Formulierontwerp




```
Dit is mijn eerste dBASE-formulier.
** END HEADER -- deze regel niet verwijderen*
* gemaakt op 13-08-94
LOCAL f
f = NEW HALLOFORM()
f.Open()
CLASS HALLOFORM OF FORM
   this.Minimize = .F.
   this.Height =         10
   this.Left =          9.5
   this.Top =          5.5293
   this.Text = "Mijn eerste dBASE-formulier"
   this.Width =         58
   this.ColorNormal = "N/GB"
   this.MousePointer =          1
   this.Maximize = .F.
   DEFINE TEXT TEKST1 OF THIS;
       PROPERTY;
         Height          2,;
         Left          8,;
         FontSize         24,;
         Top          2,;
         Text "Hallo wereld!",;
         Width         38,;
         FontItalic .T.,;
         ColorNormal "N/GB",;
         Border .F.
   DEFINE PUSHBUTTON Knop1 OF THIS;
       PROPERTY;
         Height          3,;
         Left         22,;
         OnClick CLASS::Knop1_ONCLICK,;
         FontSize         10,;
         Top          5,;
         Text "Tot ziens",;
         Width         13,;
         ColorNormal "N/W",;
         Default .T.
   Procedure Knop1_OnClick
   DO WHILE form.Height > 0 .and. form.Width > 0
     form.Tekst1.Text = "Tot ziens!"
     form.Height = form.Height - 1
     form.Width = form.Width - 1
     form.Top = form.Top + .5
     form.Left = form.Left + .5
     DO GeluidMaken
   ENDDO
   form.Close()
   RETURN
ENDCLASS
PROCEDURE GeluidMaken
   ? CHR(7)
RETURN
```

- Formulierontwerp genereert letterlijke waarden voor alle kenmerkinstellingen. Als bijvoorbeeld een uitdrukking zoals DATE() is gebruikt in de oorspronkelijke code om een datumveld te initialiseren, wordt dit veld ingesteld op een letterlijke reeks met de huidige datum in het .WFM-bestand. Als de methoden NextRow() of NextCol() zijn gebruikt in de oorspronkelijke code om de kenmerken Top of Left in te stellen voor stuurelementen, worden deze kenmerken ingesteld op letterlijke coördinaten in het .WFM-bestand.

In Afbeelding 16.2 ziet u hoe de code in een .WFM-bestand is ingedeeld.

Werken met menu-ontwerpcode

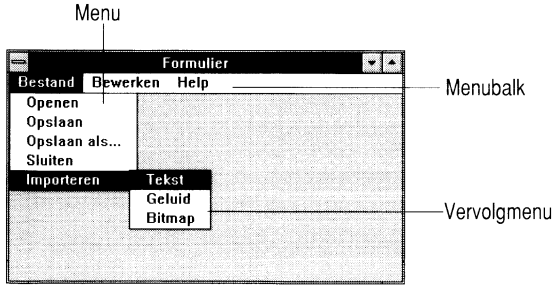
Met Menu-ontwerp kunt u een compleet menu-systeem voor een formulier maken, met een menubalk, menu's en vervolgmenu's. Elk menu-object heeft ingebouwde kenmerken waarmee sneltoetsen, ingeschakelde opties, lichter gekleurde opties en andere menu-eigenschappen kunnen worden ingesteld. U kunt code rechtstreeks koppelen aan menu-objecten met de actie OnClick, zodat u het gehele menu of een deel van het menu gemakkelijk opnieuw kunt gebruiken in een ander formulier.

Menu-ontwerp genereert een .MNU-bestand, dat u kunt koppelen aan een formulier door het .MNU-bestand op te geven bij het formulierkenmerk MenuFile.

dBASE voor Windows ondersteunt de horizontale menubalken (die zijn gemaakt met de commando's DEFINE MENU en DEFINE PAD) en popup-menu's (die zijn gemaakt met de commando's DEFINE POPUP en DEFINE BAR) van dBASE IV, om compatibel te blijven met laatstgenoemde versie. Als u menu's wilt maken in dBASE voor Windows, gebruikt u de menu-objecten die worden beschreven in deze sectie. Zie Appendix B voor meer informatie over het converteren van menu's uit dBASE IV naar dBASE voor Windows.

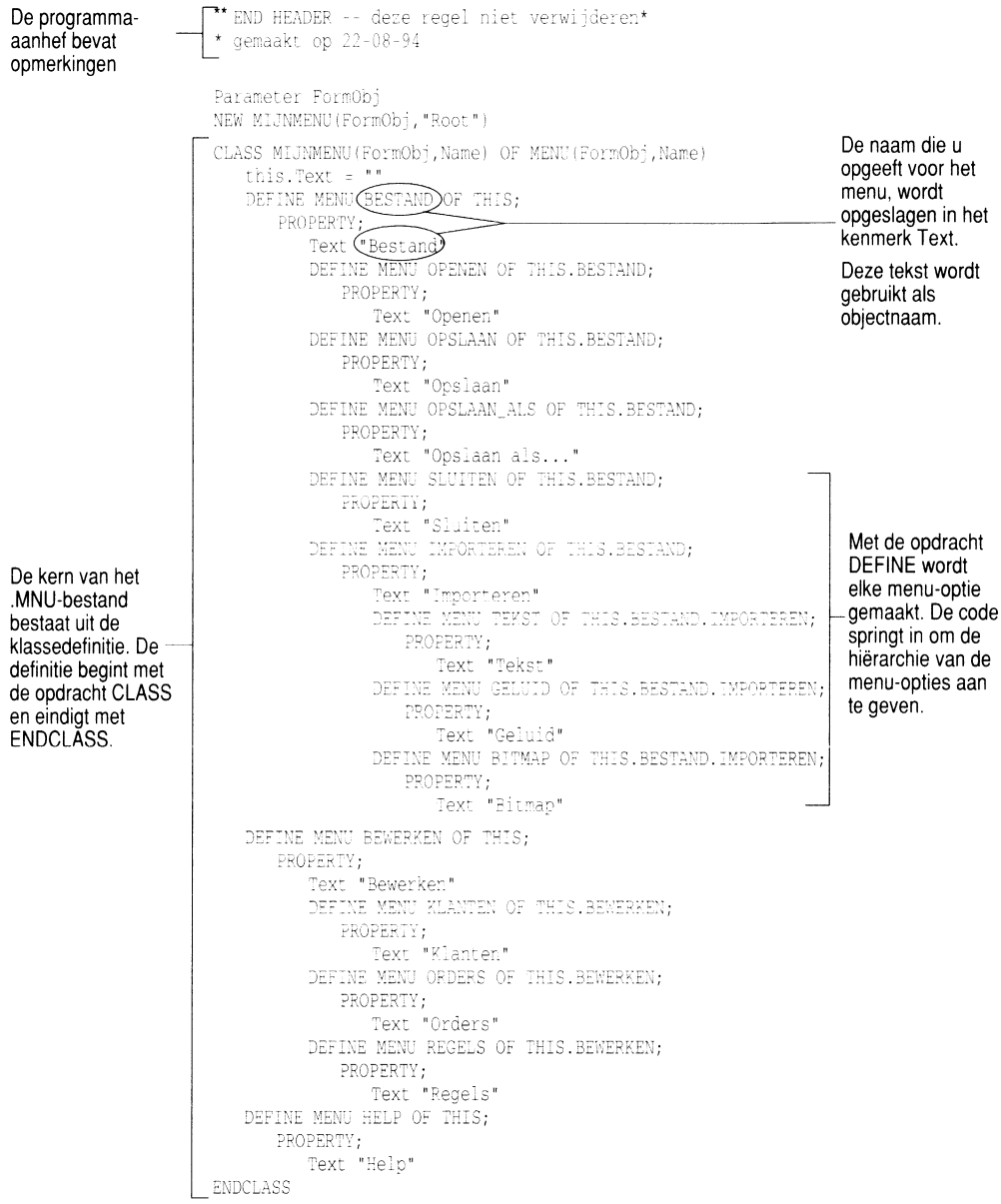
In Afbeelding 16.3 ziet u een voorbeeld van een menusysteem voor een formulier.

Afbeelding 16.3 Voorbeeldmenu's



Met de volgende code, die is gegenereerd door Menu-ontwerp, worden de menu's in Afbeelding 16.3 gemaakt.

Afbeelding 16.4 Uitleg bij een .MNU-bestand van Menu-ontwerp



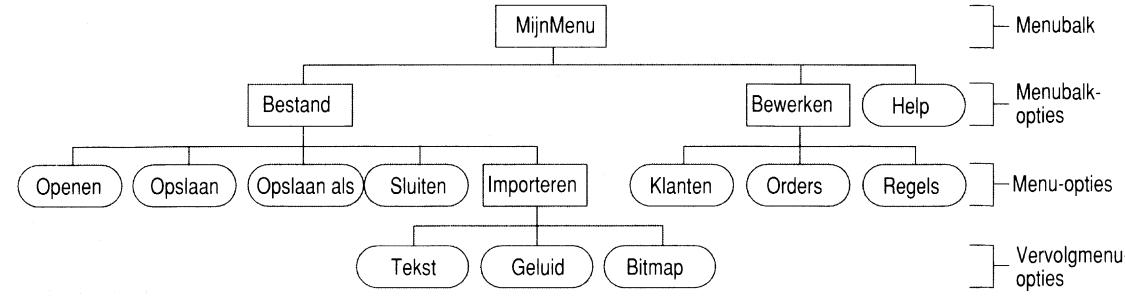

```
** END HEADER -- deze regel niet verwijderen*
* gemaakt op 22-08-94

Parameter FormObj
NEW MIJNMENU(FormObj,"Root")

CLASS MIJNMENU(FormObj,Name) OF MENU(FormObj,Name)
    this.Text = ""
    DEFINE MENU BESTAND OF THIS;
        PROPERTY;
            Text "Bestand"
            DEFINE MENU OPENEN OF THIS.BESTAND;
                PROPERTY;
                    Text "Openen"
            DEFINE MENU OPSLAAN OF THIS.BESTAND;
                PROPERTY;
                    Text "Opslaan"
            DEFINE MENU OPSLAAN_ALS OF THIS.BESTAND;
                PROPERTY;
                    Text "Opslaan als..."
            DEFINE MENU SLUITEN OF THIS.BESTAND;
                PROPERTY;
                    Text "Sluiten"
            DEFINE MENU IMPORTEREN OF THIS.BESTAND;
                PROPERTY;
                    Text "Importeren"
                    DEFINE MENU TEKST OF THIS.BESTAND.IMPORTEREN;
                        PROPERTY;
                            Text "Tekst"
                    DEFINE MENU GELUID OF THIS.BESTAND.IMPORTEREN;
                        PROPERTY;
                            Text "Geluid"
                    DEFINE MENU BITMAP OF THIS.BESTAND.IMPORTEREN;
                        PROPERTY;
                            Text "Bitmap"
    DEFINE MENU BEWERKEN OF THIS;
        PROPERTY;
            Text "Bewerken"
            DEFINE MENU KLANTEN OF THIS.BEWERKEN;
                PROPERTY;
                    Text "Klanten"
            DEFINE MENU ORDERS OF THIS.BEWERKEN;
                PROPERTY;
                    Text "Orders"
            DEFINE MENU REGELS OF THIS.BEWERKEN;
                PROPERTY;
                    Text "Regels"
    DEFINE MENU HELP OF THIS;
        PROPERTY;
            Text "Help"
ENDCLASS
```

Dit model voor menu-opbouw is gebaseerd op de hiërarchie en het container-concept van menu-objecten, niet op basis van het type menu. U definieert dus niet expliciet menubalken, menu's of vervolgmenu's. U bouwt een hiërarchie van menu-objecten op, waarin elk menu-object container is van een ander menu-object of een actie uitvoert. Net

als een formulier stuurelementen bevat, bevatten menu-objecten andere menu-objecten. Het niveau van een menu-object in de hiërarchie bepaalt waar het menu verschijnt.

In Afbeelding 16.5 ziet u de menuhiërarchie voor het formulier in Afbeelding 16.3. Elk knooppunt in de hiërarchie dat leidt naar een ander knooppunt, is een menu-object dat container is van een ander menu-object. Knooppunten aan het uiteinde van vertakkingen hebben een actie-afhandelingsroutine die is toegewezen aan de actie OnClick.

Afbeelding 16.5 Hiërarchie voorbeeldmenu



= Menu-object dat een ander menu-object inhoudt

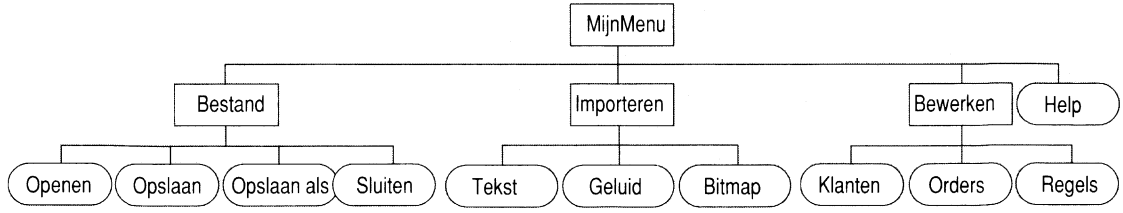
= Menu-object dat een actie start

U kunt de opbouw van een menusysteem gemakkelijk wijzigen door de container-status van een menu-object te wijzigen. De optie OF van het commando DEFINE geeft de container-status aan. In Afbeelding 16.5 is het menu Importeren bijvoorbeeld een optie van het menu Bestand. Als u Importeren wilt wijzigen van een vervolgmenu in een menubalkmenu, wijzigt u als volgt de optie OF van `Form1.MijnMenu.Bestand` in `Form1.MijnMenu`:

```
DEFINE MENU Importeren OF Form1.MijnMenu PROPERTY Text "Importeren"
```

In Afbeelding 16.6 ziet u de nieuwe menuhiërarchie.

Afbeelding 16.6 Nieuwe menuhiërarchie



= Menu-object dat een ander menu-object inhoudt

= Menu-object dat een actie start

dBASE kent geen beperkingen voor het aantal vervolgmenu's of menuniveaus. Een goed ontworpen menusysteem bevat echter niet te veel vervolgmenu's. Als u een

menuhiërarchie definieert met weinig niveaus, kunnen gebruikers acties uitvoeren met een minimumaantal stappen.

In Tabel 16.1 ziet u een aantal menukenmerken waarmee u menu-eigenschappen kunt instellen.

Tabel 16.1 Samenvatting van menukenmerken

Kenmerk	Eigenschap
Checked	Hiermee voegt u een vinkje toe naast de tekst van een menu of menu-optie.
Enabled	Hiermee geeft u op of een menu of menu-optie beschikbaar is. Als u Enabled instelt op .F. (onwaar), wordt de tekst lichter gekleurd weergegeven.
ShortCut	Hiermee geeft u een sneltoets op waarmee gebruikers de menu-optie kunnen kiezen. Met sneltoetsen (ook wel *toegangstoetsen* genoemd) kunt u snel toegang krijgen tot een menu-optie via het toetsenbord. U kunt bijvoorbeeld *Ctrl+A* opgeven bij ShortCut voor de menu-optie "Afsluiten zonder opslaan".
Separator	Hiermee stelt u een menu-optie in als scheidingslijn. Een scheidingslijn verschijnt als een horizontale lijn zonder tekst. U kunt een scheidingslijn niet kiezen of de focus geven. U gebruikt scheidingslijnen om het begin aan te geven van een groep gerelateerde menu-items.

U kunt de menu's in een formulier wijzigen als het menu is geopend. U kunt bijvoorbeeld de beschikbare menu-opties wijzigen op basis van het geselecteerde stuurelement. In het volgende voorbeeld ziet u actie-afhandelingsroutines voor de kenmerken OnGotFocus en OnLostFocus van een bladerobject. Wanneer het bladerobject de focus heeft, wordt het vooraf gedefinieerde menu Bewerken ingeschakeld. Als het bladerobject niet langer de focus heeft, wordt het menu lichter gekleurd weergegeven.

```
PROCEDURE BladerMenu                    && Toewijzen aan OnGotFocus van bladerobject
   form.root.Bewerken.Enabled = .T.
RETURN

PROCEDURE GeenBladerMenu                && Toewijzen aan OnLostFocus van bladerobject
   form.root.Bewerken.Enabled = .F.
RETURN
```

Coördinatenvlak

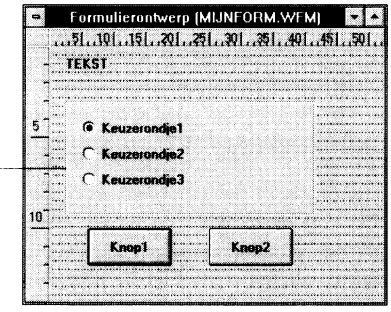
Wanneer u formulieren visueel opmaakt met Formulierontwerp, hoeft u niet na te denken over de coördinaten waarop u stuurelementen plaatst. U plaatst de stuurelementen gewoon waar u wilt, en Formulierontwerp genereert de coördinaten. Als u echter code schrijft om formulierobjecten te maken, moet u de coördinaten opgeven.

Elk formuliervenster heeft een bijbehorend font dat het formaat bepaalt van het *coördinatenvlak* waarmee u objecten plaatst in het formulier. U geeft het font op bij het kenmerk ScaleFontName. Als u ScaleFontName niet instelt, wordt het coördinatenvlak op het font MS Sans Serif font. Dit betekent dat de rijhoogte wordt bepaald door de regelhoogte van het ingestelde font en de kolombreedte wordt bepaald door de tekenbreedte van het ingestelde font. Zelfs als u een verschillend font opgeeft voor elk

stuurelement in het formulier, worden de coördinaten voor de plaatsing van stuurelementen (de kenmerken Top, Left, Height en Width) gebaseerd op het font dat u hebt opgegeven bij ScaleFontname.

In Afbeelding 16.7 ziet u hoe stuurelementen met verschillende fonts en formaten zijn geplaatst op het coördinatenvlak van het formulier.

Afbeelding 16.7 Het coördinatenvlak



Het kenmerk Left van een stuurelement geeft het aantal kolommen aan vanaf de linkermarge. Het kenmerk Top van een stuurelement geeft het aantal rijen aan vanaf de bovenkant van het formulier.

Proportionele fonts, zoals Helvetica of Times, hebben tekens met verschillende breedte (een *i* neemt minder ruimte in beslag dan een *w*). Als u een proportioneel font opgeeft bij het formulierkenmerk ScaleFontName, wordt de *gemiddelde* tekenbreedte gebruikt als kolombreedte. U kunt daarom gemakkelijker uw formuliercoördinaten bepalen met een niet-proportioneel font.

De rijhoogte en kolombreedte van een coördinatenvlak vormen samen een *tekeneenheid*. Bij kenmerken zoals Height en Width geeft u tekeneenheden op om het objectformaat en de objectpositie te bepalen. Het kenmerk Height geeft dus de hoogte aan in tekeneenheden, bijvoorbeeld:

```
MijnForm.MijnObj.Height = 2.4   && 2.4 tekeneenheden
```

De werkelijke hoogte van het object hangt af van het aantal tekeneenheden *en* het formaat van de tekeneenheden.

Als u de plaatsing van stuurelementen zeer nauwkeurig wilt bepalen, kunt u decimalen opgeven bij tekeneenheden, bijvoorbeeld:

```
DEFINE PUSHBUTTON OK OF MijnForm AT 3.5, 1.5 PROPERTY TEXT "OK"
```

Relatieve adressering

Relatieve adressering is een algemene methode in dBASE DOS om objecten te plaatsen die zijn gemaakt met het commando @...SAY...GET. U geeft hierbij coördinaten op in verhouding tot het vorige weergegeven object.

Formulierobjecten hebben de methoden NextRow() en NextCol() om relatieve adressering te kunnen toepassen in dBASE voor Windows. NextRow() resulteert in de

volgende beschikbare rij op het coördinatenvlak ten opzichte van het vorige weergegeven stuurelement en NextCol() resulteert in de volgende beschikbare kolom. De volgende stuurelementen worden bijvoorbeeld correct uitgelijnd, ongeacht de opgegeven fonts:

```
DEFINE TEXT T1 OF MijnForm AT 1,2
      PROPERTY Text "Eerste regel", FontName "Times", FontHeight 24
DEFINE TEXT T1 OF MijnForm AT MijnForm.NextRow()+1,2
      PROPERTY Text "Tweede regel", FontName "Helvetica", FontHeight 14
DEFINE TEXT T1 OF MijnForm AT MijnForm.NextRow()+1,2
      PROPERTY Text "Derde regel", FontName "Arial", FontHeight 12
DEFINE TEXT T1 OF MijnForm AT ROW(),MijnForm.NextCol()
      PROPERTY Text "Nog een object op de derde regel"
```

Deel IV

Tabellen

In dit deel komen aspecten van het programmeren met tabellen aan bod.

Is dit uw eerste kennismaking met programmeren in dBASE, lees dan eerst Hoofdstuk 17 tot en met 20 over de grondbeginselen van het werken met tabellen.

Dit deel bevat de volgende hoofdstukken:

- Hoofdstuk 17, "Werken met tabellen"
- Hoofdstuk 18, "Werken met records en velden"
- Hoofdstuk 19, "Samenhang en volgorde van records"
- Hoofdstuk 20, "Recordinformatie zoeken en samenvatten"
- Hoofdstuk 21, "Programmeren voor een netwerkomgeving"
- Hoofdstuk 22, "Werken met transacties"
- Hoofdstuk 23, "Werken met niet-dBASE tabellen"

Hoofdstuk

17

Werken met tabellen

In dBASE voor Windows worden gegevens voornamelijk opgeslagen in tabellen. In eerdere versies van dBASE komen tabellen overeen met *databasebestanden*. In dBASE voor Windows wordt één gegevensbestand een tabel genoemd, terwijl een verzameling van tabellen en andere bijbehorende bestanden een database wordt genoemd. De bestanden blijven dezelfde als in dBASE IV, alleen de terminologie is veranderd.

In dit hoofdstuk worden de belangrijkste taken en methoden behandeld die u nodig hebt om in dBASE-taal tabellen te maken en gebruiken. Zie het *Handboek* voor informatie over tabellen maken met het menu en de ontwerpvensters van dBASE.

U kunt dBASE-, Paradox- en SQL-tabellen direct openen in dBASE voor Windows. (U kunt SQL-tabellen alleen openen als Borland SQL Link is geïnstalleerd.) De commando's en de functies in dit hoofdstuk gelden voor de bewerking van al deze typen tabellen. Als dit niet het geval is, wordt het vermeld. (Zie Hoofdstuk 23 voor meer informatie over de specifieke verschillen in het gebruik van commando's en functies van dBASE als u werkt met Paradox- en SQL-tabellen.) U kunt werken met andere gegevensbestanden door deze te importeren in een dBASE-tabel. Zie de commando's IMPORT en APPEND FROM in *Commando's en functies* voor meer informatie.

Tabellen maken

U kunt dBASE- (.DBF), Paradox- (.DB) en SQL-tabellen maken in dBASE voor Windows. U maakt een tabel door de veldtypen van de tabel te benoemen en te definiëren. U geeft voor elk veld de veldnaam, het type en de lengte op. Bovendien geeft u voor elk veld op of dit een indexlabel bevat. (Voor Paradox-tabellen definieert u een primaire index.) Alle velddefinities bij elkaar vormen de tabelstructuur.

U kunt een nieuwe tabel maken door zelf alle veldbeschrijvingen op te geven. U kunt echter ook de structuur van een bestaande tabel kopiëren, de veldbeschrijvingen wijzigen en de structuur onder een nieuwe tabelnaam opslaan.

U gebruikt CREATE als u een nieuwe tabel wilt maken. Met CREATE wordt Tabelontwerp geactiveerd. Hierin kunt u de velden van een tabel interactief definiëren. Zie het *Handboek* voor meer informatie over Tabelontwerp.

Als u een tabel wilt maken waarvan de structuur is gebaseerd op die van een bestaande tabel, kunt u COPY STRUCTURE gebruiken of CREATE...FROM in combinatie met COPY STRUCTURE EXTENDED. Met CREATE...FROM stelt u een tabel samen op basis van de velddefinities afkomstig uit een tabelbeschrijvingsbestand. U maakt een tabelbeschrijvingsbestand door de structuur van een geopende tabel te kopiëren met COPY STRUCTURE EXTENDED. In het volgende voorbeeld ziet u hoe u dit doet:

```
USE MODELTBL                    && Tabel openen voor kopiëren
COPY TO TBLDESC STRU EXTE       && Tabelstructuur kopiëren
USE TBLDESC                     && Tabelbeschrijvingsbestand openen
BROWSE                          && Veldbeschrijvingen wijzigen
CREATE NEWTABLE FROM TBLDESC    && Nieuwe tabel maken
```

U hebt in dBASE ook de beschikking over het commando CREATE...STRUCTURE EXTENDED, waarmee u een tabel maakt op basis van de structuur van de huidig geopende tabel.

Tabellen openen en sluiten

Als u gegevens in een tabel wilt bewerken, opent u eerst de tabel met het commando USE..Vervolgens gebruikt u een van de commando's waarmee u gegevens in de tabel kunt bewerken.

Als u klaar bent met de bewerking van de tabel, kunt u deze sluiten. Een geopende tabel neemt geheugen in beslag, waardoor dBASE onnodig trager kan worden uitgevoerd als u de tabel niet meer nodig hebt. Geef USE op zonder tabelnaam als u een tabel in het huidige of het opgegeven werkgebied wilt sluiten. In het volgende voorbeeld ziet u hoe u tabellen opent en sluit met USE:

```
SELECT 1                    && Werkgebied 1 actief maken
USE NAMEN                   && Tabel NAMEN.DBF openen
USE                         && Tabel NAMEN.DBF sluiten
USE NAMEN IN 3              && NAMEN.DBF openen in werkgebied 3
USE                         && NAMEN.DBF is nog steeds geopend
USE IN 3                    && NAMEN.DBF is nu gesloten
```

De geopende tabel moet zich in de huidige directory bevinden, behalve als u een pad voor de tabelnaam plaatst. Bijvoorbeeld:

```
USE C:\BOEKH\NAMEN          && Pad voor de tabel opgeven
```

U kunt nog meer opties bij USE gebruiken. Met NOUPDATE wordt de tabel alleen voor lezen geopend. Met EXCLUSIVE zorgt u dat in een gemeenschappelijke omgeving andere gebruikers geen toegang hebben tot de tabel. Met AGAIN opent u een tweede instantie van een tabel in een ander werkgebied. Zie *Commando's en functies* voor een compleet overzicht van de opties en een beschrijving van USE.

De instelling **Sessies** in het dialoogvenster **Kenmerken | Bureaublad | Bestanden**heeft ook invloed op de omgevingsinstellingen voor de tabellen die u opent en sluit vanuit het

commandovenster. Zie "Werken met sessies" in Hoofdstuk 21 en het commando CREATE SESSIONS in *Commando's en functies* voor meer informatie.

Hoewel dBASE automatisch alle geopende tabellen sluit zodra u de toepassing afsluit, is het toch verstandig om de tabellen zelf te sluiten. Met CLOSE ALL en CLOSE TABLES sluit u elke tabel in elk werkgebied. In applicaties nemen ontwikkelaars een dergelijk commando vaak op onmiddellijk voor het punt waar de applicatie wordt afgesloten.

Werken met databases

Als u een tabel maakt of opent, kunt u ook de lokatie opgeven die deze in een database heeft. In een database wordt de lokatie bepaald waar de tabel wordt opgeslagen in een SQL-database of in een directory op de vaste schijf of de file-server. (*Databases* of *database-aliassen* worden gemaakt met het configuratieprogramma IDAPI. Zie *Aan de slag* voor meer informatie.)

U opent een database met OPEN DATABASE en sluit deze met CLOSE DATABASE. Als u meerdere databases tegelijk definieert of opent, selecteert u de huidige database met het commando SET DATABASE. Als u een andere database wilt opgeven dan de huidige database, plaatst u de naam van de database (tussen dubbele punten) voor de tabelnaam.

```
OPEN DATABASE Boekh              && Eerste database openen
OPEN DATABASE Verkoop            && Tweede database openen
SET DATABASE TO Boekh            && Huidige database instellen op Boekh
USE NAMEN                        && Tabel openen in huidige database
USE :Verkoop:Klanten             && Tabel openen in database Verkoop
CLOSE DATABASES                  && Alle geopende databases sluiten
```

Als u een database opent met OPEN DATABASE, SET DATABASE of USE zonder een wachtwoord op te geven, verschijnt een dialoogvenster. Hierin typt u de informatie, zoals de gebruikersnaam en het wachtwoord, die nodig is om de opgegeven database te openen. Vervolgens kunt u in deze database bestaande tabellen openen of nieuwe tabellen maken of kopiëren.

Tabelnamen en -typen opgeven

Als u eentabel wilt gebruiken, geeft u de letterlijke tabelnaam op of een tekenuitdrukking die evalueert tot de tabelnaam. Als u een bestandsnaam opslaat in een geheugenvariabele, plaatst u de variabelenaam tussen haakjes in het commando USE. De variabele wordt dan eerst geëvalueerd, waarna het resultaat aan het commando wordt doorgegeven.

```
USE NAMEN                        && Tabel Namen openen
USE "NAMEN"                      && Tabel Namen openen
cFileName = "PLAATSEN"           && Tabelnaam opslaan in een variabele
USE (cFileName)                  && Tabel Plaatsen openen
USE &cFileName                   && Tabel openen met macro-vervangingsoperator
```

Als u geen bestandsextensie opgeeft, wordt bij de uitvoering van USE het tabeltype gebruikt dat u bij SET DBTYPE hebt opgegeven. DBTYPE is standaard ingesteld op dBASE. U kunt DBTYPE instellen op dBASE of PARADOX.

Als u een ander tabeltype wilt openen, typt u de bestandsextensie achter de bestandsnaam. U kunt ook bij USE de optie TYPE gebruiken om de instelling bij DBTYPE te vervangen. De code blijft dan overzichtelijker en u hoeft geen bestandsextensie op te geven.

```
USE FACTUREN TYPE PARADOX       && Paradox-tabel openen
SET DBTYPE TO PARADOX           && Paradox instellen als het standaardtabeltype
USE NABESTEL                    && NABESTEL.DB openen
USE NAMEN.DBF                   && dBASE-tabel openen
USE GOEDEREN TYPE DBASE         && dBASE-tabel openen
```

Naar tabellen verwijzen: aliassen

Een *alias* is een andere naam voor een geopende tabel en kan maximaal 32 tekens bevatten. U kunt een alias voor een tabel definiëren als u de tabel opent met USE. Nadat u een alias hebt gedefinieerd, kunt u de aliasnaam gebruiken in plaats van de werkelijke tabelnaam als u in een commando of een functie naar deze tabel verwijst. Een alias maakt de broncode beter leesbaar, omdat de vaak cryptische bestandsnamen dan worden vervangen door een begrijpelijke naam.

```
SELECT A
USE REK94JAN ALIAS Rekeningen     && Tabel waarvan naam maandelijks wordt gewijzigd
? ALIAS()                         && "Rekeningen" als resultaat geven
? Wissel_datum                    && Veld weergeven in het huidige werkgebied
? A->Wissel_datum                 && Hetzelfde veld weergeven door letter van het
                                  && werkgebied op te geven
? Rekeningen->Wissel_datum        && Hetzelfde veld weergeven door alias van de tabel
                                  && op te geven
```

Verwar aliasnamen van tabellen niet met *database-aliassen*. Een database-alias geeft de lokatie aan van een tabel. Dit kan zowel een SQL-database zijn als een directory op de vaste schijf of de file-server. U maakt een database-alias met het configuratieprogramma IDAPI. Zie *Aan de slag* voor meer informatie over het gebruik van het configuratieprogramma IDAPI.

Werken met meerdere tabellen: werkgebieden

U kunt tegelijkertijd met gegevens uit verschillende tabellen werken. U maakt dan gebruik van werkgebieden. Een *werkgebied* is een lokatie in het geheugen die is gereserveerd voor een geopende tabel en de bijbehorende bestanden, zoals memo-, binaire en indexbestanden. Elke tabel die u opent heeft een eigen werkgebied nodig. Er kunnen maximaal 225 werkgebieden tegelijk zijn geopend.

De commando's en de functies waarmee u gegevens inleest, gelden voor het huidige werkgebied, tenzij u een ander werkgebied opgeeft. Als u bijvoorbeeld twee tabellen hebt geopend, hangt het resultaat van het commando LIST af van het huidige werkgebied. Als het eerste werkgebied actief is, geeft u met LIST de records van de eerste tabel weer. Als het tweede werkgebied actief is, worden de records van de tweede tabel weergegeven.

U gebruikt SELECT om het huidige werkgebied in te stellen. U kunt op de volgende manieren naar een werkgebied verwijzen:

- Met een nummer (1 tot en met 225)
- Met een letter (A voor werkgebied 1, B voor werkgebied 2 enzovoort tot en met J)
- Met de aliasnaam van de geopende tabel in een werkgebied

```
SELECT 1                    && Werkgebied 1 actief maken
USE NAMEN                   && NAMEN.DBF openen in werkgebied 1
LIST                        && Records van NAMEN.DBF weergeven
SELECT B                    && Werkgebied 2 weergeven
USE PLAATSEN ALIAS Stad     && PLAATSEN.DBF openen met alias Stad
LIST                        && Records van PLAATSEN.DBF weergeven
SELECT Namen                && Werkgebied 1 selecteren
SELECT Stad                 && Werkgebied 2 selecteren
```

Het gebruik van tabelnamen of aliasnamen in het commando SELECT biedt een voordeel ten opzichte van het gebruik van het nummer of de letter van een werkgebied. U hoeft SELECT dan niet te herschrijven als u het werkgebied wijzigt. Als u een tabel opent in een nieuw werkgebied, worden het nummer en de letter van het werkgebied wel gewijzigd, terwijl de tabel of de alias niet wordt gewijzigd.

De optie IN van het commando USE zorgt dat er in een werkgebied van dBASE een tabel wordt geopend of gesloten zonder dat dit werkgebied actief wordt. U kunt ook andere dBASE-commando's waarvoor toegang tot gegevens is vereist, zoals GO, SKIP en LIST STRUCTURE, gebruiken met de optie IN om records in een ander werkgebied te bewerken.

```
SELECT 1                    && Werkgebied 1 actief maken
USE NAMEN                   && NAMEN.DBF openen in werkgebied 1
USE REGIO IN 3 ORDER Regio  && REGIO.DBF openen in werkgebied 3
                            && Werkgebied 1 is nog steeds actief
SELECT C                    && Werkgebied 3 is nu actief
GO 10                       && Naar record 10 in REGIO.DBF
GO 20 IN A                  && Naar record 20 in NAMEN.DBF
```

Een aantal dBASE-functies, zoals EOF(), FOUND() en SEEK(), beschikt eveneens over opties waarmee u het werkgebied kunt opgeven.

```
SELECT 1                    && Werkgebied 1 actief maken
SEEK("NW", 3)               && Regio "NW" in werkgebied 3 zoeken
                            && Index vereist voor veld waarnaar wordt gezocht
IF FOUND(3)                 && Bepaalt wat er gebeurt als het record wordt gevonden
 :
ENDIF
```

Als u een tabel hebt geopend en wilt werken met gegevens uit een tweede tabel, wilt u deze tabel waarschijnlijk openen in het volgende beschikbare werkgebied. De functie SELECT() resulteert in het nummer van het volgende beschikbare werkgebied.

```
USE NAMEN                   && NAMEN.DBF openen
USE PLAATSEN IN SELECT()    && Openen in het volgende beschikbare werkgebied
USE REGIO IN SELECT()       && Openen in het volgende beschikbare werkgebied
```

Gebruik de volgende commando's en functies om informatie te krijgen over werkgebieden:

- LIST STATUS. Toont het werkgebied waarin elke tabel is geopend en geeft aan welk werkgebied actief is.
- DBF(). Resulteert in de naam van de tabel die is geopend in een bepaald werkgebied.
- WORKAREA(). Resulteert in het nummer van het huidige werkgebied.

Tabellen hernoemen

U kunt in dBASE de naam van een tabel op schijf optwee manieren wijzigen:

- RENAME. Wijst een nieuwe naam toe aan een willekeurig bestand. U geeft de bestandsextensie op met RENAME. Als de tabel bijbehorende .MDX- of .DBT-bestanden heeft, moet u deze bestanden eveneens hernoemen.
- RENAME TABLE. Hernoemt een tabel en elk bijbehorend bestand met dezelfde naam, zoals produktie-indexbestanden (.MDX) en memobestanden (.DBT). Als u RENAME TABLE gebruikt voor Paradox-tabellen, wordt de volledige tabelfamilie hernoemd, inclusief indexbestanden en .VAL-bestanden. U gebruikt bij RENAME TABLE de optie TYPE om aan te geven welk tabeltype, dBASE of Paradox, u wilt hernoemen.

```
RENAME NAMEN.DBF TO MENSEN.DBF              && NAMEN.DBF hernoemen in MENSEN.DBF
RENAME TABLE NAMEN TO MENSEN TYPE DBASE     && Tabel NAMEN hernoemen samen met alle
                                            && bijbehorende bestanden met dezelfde naam
```

Als u een tabel hernoemt, moet u tevens alle formulieren, rapporten, labels en programmabestanden hernoemen die naar de oude tabelnaam verwijzen.

Tabellen kopiëren

dBASE kent diverse opties waarmee u een tabel kunt kopiëren of gegevens kunt kopiëren naar een tabel in een database. U kunt kopieën maken van een volledige tabel, een bepaald aantal records of alleen van de structuur zonder records. U kunt ook afzonderlijke DOS-bestanden kopiëren door de bestandsextensie van een bepaald bestand op te geven.

- COPY kopieert een volledige tabel of de geselecteerde records en velden. U kunt alleen een geopende tabel kopiëren. Gebruik COPY met de optie TYPE als u een tabel van het ene type wilt kopiëren naar het andere (dBASE of Paradox). Als u een .DBF-tabel naar een andere .DBF-tabel kopieert, wordt met COPY automatisch een memobestand gemaakt voor de nieuwe tabel als u tevens de memovelden kopieert. Als een index actief is, worden de records volgens de indexvolgorde naar de nieuwe tabel gekopieerd. Er wordt echter alleen een .MDX-bestand gemaakt voor de nieuwe tabel als u de optie WITH PRODUCTION opgeeft.

- COPY...WITH PRODUCTION kopieert een dBASE-tabel naar een andere dBASE-tabel met een andere naam en kopieert daarbij tevens de bijbehorende memobestanden (.DBT) en produktie-indexbestanden (.MDX).
- COPY STRUCTURE maakt een lege kopie van een tabel, dat wil zeggen dat de records niet worden gekopieerd. Open de tabel waarvan u de structuur wilt kopiëren, voordat u het commando COPY STRUCTURE opgeeft.
- COPY TABLE kopieert een tabel met alle bijbehorende bestanden, zoals produktie-indexbestanden (.MDX) en memobestanden (.DBT). Gebruik COPY TABLE als u een tabel snel wilt dupliceren. Als u COPY TABLE gebruikt voor Paradox-tabellen, wordt de volledige tabelfamilie gekopieerd, inclusief indexbestanden en .VAL-bestanden. U hoeft de tabel niet te openen voordat u deze kopieert.
- COPY FILE kopieert elk willekeurig bestand waarvan u de bestandsextensie opgeeft. Sluit de tabel voordat u deze kopieert met het commando COPY FILE. .DBT- en .MDX-bestanden worden niet met de tabel gekopieerd met COPY FILE. U moet deze bestanden afzonderlijk kopiëren.

```
COPY FILE BOEKH.PRG TO OUDBOEKH.PRG      && Bestand BOEKH.PRG kopiëren
USE NAMEN
COPY TO MENSEN TYPE PARADOX              && NAMEN.DBF kopiëren naar een Paradox-tabel
COPY STRUCTURE TO KLANTEN                && Tabelstructuur van NAMEN.DBF kopiëren
COPY TABLE NAMEN TO MAILING              && Tabel NAMEN.DBF en bijbehorende
                                         && bestanden kopiëren naar andere dBASE-tabel
```

Tabelstructuur wijzigen

Mogelijk wilt u de structuur van een tabel wijzigen nadat u deze hebt gemaakt. Zelfs als u gegevens in de tabel hebt ingevoerd, kunt u de tabelstructuur nog wijzigen. U kunt de volgende wijzigingen aanbrengen:

- Velden toevoegen
- Velden verwijderen
- Veldlengte of gegevenstype wijzigen
- Volgorde van velden wijzigen

Met LIST STRUCTURE geeft u informatie weer over de tabelstructuur.

MODIFY STRUCTURE, waarmee u Tabelontwerp activeert, is het primaire commando om een tabelstructuur te maken. In Tabelontwerp kunt u de velddefinities interactief wijzigen. Zie het *Handboek* voor meer informatie over het gebruik van Tabelontwerp.

U kunt de volledige tabeltructuur ook wijzigen met de programmabesturing door een gewijzigde *kopie* van de tabel te maken met COPY TO STRUCTURE EXTENDED en CREATE FROM. In de sectie "Tabellen maken" in dit hoofdstuk wordt aan de hand van een voorbeeld uitgelegd hoe u deze commando's samen gebruikt.

Tabellen verwijderen

Gebruik DELETE TABLE, DELETE FILE of ERASE om een tabel definitief van een schijf te verwijderen.

Met ERASE verwijdert u een bestand van een schijf. DELETE FILE heeft hetzelfde resultaat als ERASE. Bijbehorende bestanden, zoals index- en memobestanden, worden nietverwijderd met deze commando's. U moet deze dan afzonderlijk verwijderen.

Met DELETE TABLE verwijdert u een tabel inclusief de bijbehorende bestanden met dezelfde naam, zoals .MDX- of .DBT-bestanden. De bestanden die voortkomen uit de tabel, zoals formulieren, rapporten, labels of programmabestanden, worden niet verwijderd met dit commando. Als u DELETE TABLE gebruikt voor Paradox-tabellen, wordt de volledige tabelfamilie verwijderd, inclusief .VAL-bestanden.

U moet de tabel eerst sluiten, voordat u ERASE, DELETE FILE of DELETE TABLES opgeeft om de tabel met bijbehorende bestanden te verwijderen.

```
ERASE TIJDLK.DBF              && Bestand TIJDLK.DBF verwijderen
DELETE FILE TIJDLK.DBF        && Bestand TIJDLK.DBF verwijderen
DELETE TABLE TIJDLK.DBF       && Tabel TIJDLK.DBF en alle
                              && bijbehorende bestanden verwijderen
```

Als u een tabel verwijdert, moet u alle formulieren, rapporten, labels en programmabestanden bijwerken die afhankelijk zijn van informatie uit de tabel.

Hoofdstuk

18

Werken met records en velden

U kunt in dBASE gegevens in tabellen interactief weergeven en bewerken met bewerkingen op het bureaublad. Daarnaast kunt u zelf dBASE-programma's en formulieren ontwerpen om gegevens op te halen en bij te werken.

In dit hoofdstuk worden de methoden beschreven waarmee u gegevens in tabellen weergeeft en bewerkt met behulp van commando's en functies van de dBASE-taal. (Zie het *Handboek* voor informatie over het weergeven en bewerken van gegevens met de gebruikersinterface. Zie Hoofdstuk 21 van deze handleiding voor meer informatie over het bewerken van gegevens in een gemeenschappelijke omgeving.)

De commando's en de functies in dit hoofdstuk hebben dezelfde werking in Paradox-, dBASE- en SQL-tabellen, tenzij anders wordt vermeld. (Zie Hoofdstuk 23 voor meer informatie over specifieke verschillen in het gebruik van commando's en functies van dBASE bij het werken met Paradox- en SQL-tabellen.)

Werken met records

In een tabel vormenrecords het belangrijkste middel om informatie op te slaan die u later kunt opzoeken en ophalen. In de afzonderlijke velden van een record kunt u alle gegevens opslaan die kenmerkend zijn voor een bepaald record. Een tabel kan bijvoorbeeld informatie bevatten over elke afzonderlijke klant of werknemer. Elk record bevat dan velden waarin bijvoorbeeld de naam, het telefoonnummer en het adres van een bepaalde klant of werknemer zijn opgeslagen.

Records toevoegen en gegevens bewerken behoren tot de primaire bewerkingen die u uitvoert in een dBASE-programma. Voordat u deze taken echter kunt uitvoeren, moet u eerst een aantal basistechnieken kennen. U moet bijvoorbeeld weten hoe u door records navigeert en commando's kiest om gegevens toe te voegen en bij te werken. Dit hoofdstuk biedt informatie over commando's en functies waarmee u gegevens in velden en records kunt plaatsen en bijwerken. Hierbij wordt zowel het interactief gebruik beschreven als het gebruik ervan in programma's. In dit hoofdstuk worden de volgende afzonderlijke onderwerpen behandeld:

- Het commando BROWSE gebruiken
- Navigeren door records
- Velden opgeven en selecteren
- Records toevoegen
- Records bijwerken
- Records markeren en verwijderen
- Werken met AUTOMEM-variabelen
- Werken met memo-, binaire en OLE-velden

U kunt ook eigen objecten ontwerpen waarmee gegevens in een tabel interactief kunnen worden weergegeven en bewerkt. U kunt bijvoorbeeld velden op een formulier definiëren en deze met de kenmerken DataSource en DataLink koppelen aan afzonderlijke velden in een tabel. Zie deel III, "Formulieren", van deze handleiding voor meer informatie over het zelf ontwerpen van formulieren.

Commando BROWSE gebruiken

Het commando BROWSE biedt u de snelste manier om gegevens in dBASE te bewerken. U kunt dit opgeven vanuit het commandovenster of vanuit een dBASE-programma. Bijvoorbeeld:

```
USE ORDERS
BROWSE
```

dBASE-commando's die u in het commandovenster typt, hebben hetzelfde resultaat als de equivalente bewerkingen via de gebruikersinterface of de ontwerpvensters. Het commando BROWSE geeft bijvoorbeeld meerdere records weer in een venster waarvan u de grootte kunt wijzigen. Hetzelfde venster kan verschijnen als u een tabel opent met Navigator. (Met de opties van **Bureaublad** bepaalt u of de tabel wordt geopend in de modus voor bewerken in bladeropmaak, formulieropmaak of kolomopmaak.) Met BROWSE kunt u records in de tabel weergeven, gegevens in velden bewerken en nieuwe records toevoegen aan het einde van de tabel.

Commando-opties van BROWSE

Bij het commando BROWSE kunt u een aantal opties opgeven. U kunt bijvoorbeeld de optie FIELDS opgeven om velden te selecteren in de tabel die in het venster wordt weergegeven.

```
BROWSE FIELDS ORDERDAT, VERZENDDAT, BETVOORW
```

Andere commando's zoals NOAPPEND, NOEDIT en NODELETE geven het type wijziging aan dat u in een tabel kunt aanbrengen. Met FOR, WHILE en andere bereikopties selecteert u de records die in het BROWSE-venster worden weergegeven. (Zie *Commando's en functies* voor meer informatie over de opties bij het commando

BROWSE.) Daarnaast kunt u met het commando SET FIELDS de velden beperken die beschikbaar zijn met BROWSE.

Gegevens bewerken per record

Druk op *F2* of kies **Weergave | Formulieropmaak** in **Tabelrecords** als u de gegevens per record wilt weergeven en bewerken. Dit heeft hetzelfde resultaat als het commando EDIT in het commandovenster of in een dBASE-programma. U kunt *PgUp*, *PgDn* of de pijltoetsen gebruiken om de cursor te verplaatsen naar een ander veld of record in het venster.

U kunt de gegevens ook bewerken met het commando APPEND. Als u APPEND opgeeft, verschijnt een venster waarin de velden van een nieuw, leeg record worden weergegeven. Nadat u APPEND hebt opgegeven, kunt u niet alleen nieuwe records toevoegen, maar u kunt ook door de tabel navigeren om bestaande records te bewerken door de cursor te verplaatsen naar vorige records in de tabel.

```
USE ORDERS                    && Tabel ORDERS.DBF openen
APPEND                        && Venster openen met de recordaanwijzer
                              && aan het einde van de tabel
```

De optie BLANK van het commando APPEND is een programmeercommando. Daarom kunt u de tabel met dit commando niet in een bewerkvenster weergeven. Er wordt slechts een nieuw, leeg record toegevoegd aan het einde van de tabel.

Wijzigingen opslaan

Als u een record hebt toegevoegd of gewijzigd, worden de wijzigingen in de tabel opgeslagen zodra u de cursor verplaatst. De wijzigingen worden opgeslagen in een recordbuffer in het geheugen en weggeschreven naar de schijf als de recordbuffer vol is. Gebruik SET AUTOSAVE ON als u de kans op gegevensverlies verder wilt verkleinen. U zorgt dan dat elke wijziging in een record wordt weggeschreven naar de schijf.

Als u een wijziging direct wilt wegschrijven naar de schijf als SET AUTOSAVE OFF actief is, gebruikt u het commando FLUSH nadat u een record hebt toegevoegd of gewijzigd. Wijzigingen worden ook naar de schijf weggeschreven als u de commando's BROWSE, EDIT of APPEND afsluit door op *Ctrl-W* te drukken, als u het venster sluit en daarbij de wijzigingen opslaat, of als u in het menu **Bestand** de tabel opslaat en sluit.

Als u een tabel wilt sluiten zonder de wijzigingen in het huidige record op te slaan, drukt u op *Esc* of *Ctrl-Q* of kiest u in **Bestand** de opties waarmee de wijzigingen in het laatste record worden geannuleerd en de tabel wordt gesloten.

Navigeren door records

Voordat u gegevens bewerkt, bepaalt u eerst welke records u wilt wijzigen. Met de commando's die verwijzen naar afzonderlijke records en waarmee de *recordaanwijzer* wordt verplaatst, kunt u in de tabel records selecteren die u wilt bewerken. U kunt de cursor vooruit en achteruit in de tabel verplaatsen, het eerste of het laatste record activeren of de cursor naar een bepaald record verplaatsen.

U hebt in dBASE eveneens de beschikking over commando's en functies waarmee u de records kunt tellen die voldoen aan een bepaalde voorwaarde, of waarmee u het nummer, het type of de naam van velden kunt bepalen.

Recordnummers en bladwijzers

Als u een nieuw record toevoegt aan een dBASE-tabel, wordt hieraan een uniek recordnummer toegewezen. Als u de tabel bewerkt, gebruikt u de recordnummers om afzonderlijke records te identificeren en bepaalde records te selecteren, zodat u gegevens kunt weergeven, bijwerken of verwijderen. U gebruikt de recordnummers ook als u een record wilt vergrendelen voordat u het bijwerkt.

Als u een tabel voor de eerste keer opent, wordt de recordaanwijzer op het eerste record geplaatst. Vervolgens kunt u commando's en functies gebruiken om gegevens te bewerken of de cursor naar andere records te verplaatsen.

Paradox- en SQL-tabellen hebben, in tegenstelling tot dBASE-tabellen, geen vaste recordnummers. Als u gegevens van deze tabeltypen ophaalt in dBASE, wordt er een speciale dynamische markering, bladwijzer genaamd, toegewezen. Hoewel bladwijzers te vergelijken zijn met de recordnummers in dBASE-tabellen, zijn er een aantal verschillen. Zie Hoofdstuk 23 voor meer informatie.

Cursor naar een bepaald record verplaatsen

Gebruik het commando GO (of GOTO) om de recordaanwijzer naar een bepaald record in een geopende tabel te verplaatsen. Bijvoorbeeld:

```
GO 23                   && Recordaanwijzer verplaatsen naar record 23
DISPLAY                 && Huidige record, recordnummer 23, tonen
GO TOP                  && Recordaanwijzer verplaatsen naar het eerste record in tabel
? RECNO()               && Recordnummer van het huidige record tonen
GO BOTTOM               && Recordaanwijzer verplaatsen naar het laatste record in tabel
gRec = 20               && Recordnummer opslaan in een geheugenvariabele
GO gRec                 && Recordaanwijzer verplaatsen op basis van de waarde van gRec
```

U kunt GO gebruiken om de recordaanwijzer te verplaatsen in een Paradox- of SQL-tabel. U kunt hierbij echter geen numerieke waarde opgeven. U geeft dan de bladwijzer op die met de functies BOOKMARK() of RECNO() is toegewezen aan een bepaald record.

```
GO TOP                  && Recordaanwijzer verplaatsen naar het eerste record in tabel
LOCATE FOR NAME = "Jan" && Recordaanwijzer verplaatsen naar het record met
                        && de naam "Jan"
gBkmark = BOOKMARK()    && Bladwijzer opslaan in geheugenvariabele
GO TOP                  && Recordaanwijzer verplaatsen naar eerste
                        && logische record in tabel
GO gBkmark              && Recordaanwijzer verplaatsen op basis van waarde van
                        && eerder opgeslagen bladwijzer
```

Hoewel u in commando's en functies toegewezen bladwijzers kunt gebruiken in plaats van recordnummers, kunt u de waarde van de bladwijzer niet rechtstreeks weergeven.

U kunt wel uitdrukkingen maken waarin de relatieve waarde van twee bladwijzers met elkaar wordt vergeleken. Bijvoorbeeld:

```
IF BkMark1 < BkMark2        && Controleren of eerste bladwijzer voor
                            && tweede bladwijzer in tabel komt
```

Bereikopties gebruiken

Bij de meeste commando's en functies waarmee u records bewerkt, kunt u ook de optie *<bereik>* opgeven. Hiermee selecteert u de records die u wilt verwerken. U kunt bij BROWSE en EDIT bijvoorbeeld opgeven of u een afzonderlijk recordnummer, een verzameling recordnummers, ALL of REST (de resterende records in de tabel) wilt selecteren.

```
BROWSE REST                 && Resterende records in BROWSE-venster selecteren
```

Met de meeste commando's en functies van dBASE bewerkt u het huidige record, tenzij u een bepaald bereik opgeeft. Controleer echter de syntaxis van een commando of een functie, zodat u weet welke records worden bewerkt. Met BROWSE en EDIT worden bijvoorbeeld alle records in de tabel bewerkt als u geen bereik opgeeft of de opties FOR en WHILE niet opneemt in de code. Met DISPLAY wordt het huidige record weergegeven als u geen andere opties gebruikt.

Vooruit en achteruit verplaatsen

Met het commando SKIP verplaatst u de recordaanwijzer vooruit of achteruit vanaf de huidige lokatie. Bijvoorbeeld:

```
SKIP 5                      && Recordaanwijzer vijf records vooruit plaatsen
SKIP -10                    && Recordaanwijzer tien records achteruit plaatsen
```

De volgorde van de opgehaalde records wordt bestuurd door de indexen die u opgeeft als u een tabel opent met het commando USE. De volgorde van de records heeft ook invloed op de werking van de meeste overige commando's en functies waarmee u records weergeeft en bewerkt. Zie Hoofdstuk 19 voor meer informatie over indexen.

Positie van de recordaanwijzer bepalen

De functie RECNO() resulteert in de waarde van het huidige record in een dBASE-tabel. In Paradox- en SQL-tabellen resulteert de functie in de waarde van de bladwijzer die is toegewezen aan het huidige record. Met de functies BOF() en EOF() kan worden aangegeven dat u het begin of het einde van een tabel hebt bereikt.

```
USE BEDRIJF
LOCATE FOR REGIO = "NW"        && Eerste record zoeken waarbij REGIO = "NW"
gRec = RECNO()                 && Waarde van huidige recordnummer opslaan
GO TOP                         && Recordaanwijzer verplaatsen naar eerste record
DO WHILE .NOT. EOF()           && Records verwerken tot aan einde van bestand EOF()
   ? BEDRIJF, VERKTOTNU
   SKIP
ENDDO
GO gRec                        && Recordaanwijzer terugplaatsen naar waarde van gRec
```

Records tellen

Verder kunt u met de functie RECCOUNT() of het commando COUNT het aantal records in de huidige tabel bepalen. Met COUNT kunt u een voorwaarde opgeven voor de records die u wilt tellen.

```
USE BEDRIJF
Rec_cnt = RECCOUNT()           && Alle records in tabel BEDRIJF.DBF tellen
COUNT TO Qual_Cnt;             && Alle records tellen waar regio = NW
                               && (controleren op hoofd- en kleine letters)
  FOR REGIO = "NW"
? Qual_cnt/Rec_cnt             && Percentage berekenen van records
                               && waar regio = NW
```

U kunt het commando SET FILTER wel gebruiken voor records die zijn verwerkt met het commando COUNT, maar niet voor records die zijn verwerkt met de functie RECCOUNT().

Velden opgeven en selecteren

Bij veel dBASE-commando's kunt u een veldenlijst opgegeven waarmee uit de huidige tabel van een database of uit een tabel in een ander werkgebied de velden worden bepaald die u wilt weergeven of verwerken. Dit is handig als u bijvoorbeeld niet alle velden in een tabel wilt weergeven of bewerken, of als u velden van gerelateerde records uit meerdere tabellen weergeeft.

Veldenlijst opgeven

Met het commando SET FIELDS geeft u een lijst met velden op voor de commando's waarbij een veldenlijst kan worden opgenomen in de syntaxis. U kunt de veldenlijst die u hebt ingesteld met SET FIELDS TO ook inschakelen of uitschakelen met SET FIELDS ON of OFF. Met CLEAR FIELDS verwijdert u de veldenlijst die u hebt ingesteld met het commando SET FIELDS.

```
USE BEDRIJF
SET FIELDS TO BEDRIJF, REGIO    && Veldenlijst instellen met twee velden
LIST ALL FOR REGIO = "NW"       && Alle records met regio NW weergeven
```

SET FIELDS beschikt ook over opties waarmee u rekenvelden definieert en bepaalde velden instelt op alleen-lezen.

Velden gebruiken uit meerdere werkgebieden

U kunt met SET FIELDS ook velden opnemen uit meerdere tabellen die in verschillende werkgebieden zijn geopend. Gebruik de notatie *alias->veld* als u velden opgeeft van een tabel die zich niet in het huidige werkgebied bevindt.

```
SELECT 1
USE ORDERS ORDER OrderNr              && Tabel openen in werkgebied 1
SELECT 2
USE REGELS ORDER OrderNr              && Tabel openen in werkgebied 2
SET RELATION TO OrderNr INTO ORDERS   && Relatie instellen tussen REGELS en ORDERS.DBF
SET FIELDS TO ORDERS->KLANTNR,;       && Veldenlijst definiëren op basis van
                                      && één of meer tabellen
   ORDERNR, VOORRAADNR, AANTAL, TOTAAL && Velden selecteren in niet-actief werkgebied
```

Opmerking U kunt in de veldenlijst velden uit de tabellen in alle werkgebieden opnemen. Gebruik in dat geval SET RELATION om een relatie te leggen tussen de tabellen in de verschillende werkgebieden. Zie Hoofdstuk 19, "Samenhang en volgorde van records" voor informatie over relaties tussen tabellen en de bijbehorende opties.

Informatie over velden gebruiken

De functie FLDLIST() resulteert in de huidige veldenlijst die u hebt ingesteld met het commando SET FIELDS. U kunt ook de functies FIELD(), FLDCOUNT() en FLENGTH() gebruiken om informatie op te vragen over de velden in een tabel. FIELD() resulteert in de naam van een veld op basis van het veldnummer in de tabel. FLDCOUNT() resulteert in het aantal velden in de tabel en FLENGTH() in de lengte van een bepaald veld.

```
USE BEDRIJF
SET FIELDS TO BEDRIJF, PLAATS
? FLDLIST()                           && Velden uit huidige veldenlijst opvragen.
DECLARE FldArray[FLDCOUNT()]          && Array maken met overeenkomstig aantal velden
                                      && in de tabel. Array gebruiken om waarden
                                      && van en naar velden in een record te kopiëren
FOR FldPtr = 1 TO FLDCOUNT()          && Maakt lussen totdat FldPtr de waarde FLDCOUNT()
                                      && bereikt
   ? FIELD(FldPtr)                    && Veldnamen uit veldenlijst als resultaat geven
NEXT
```

Scheidingstekens bij veldnamen

U kunt geen spaties en andere speciale tekens opnemen in de naam van een veld in een dBASE-tabel. (Een veldnaam moet met een letter beginnen en kan verder letters, getallen en onderstreeptekens bevatten.) De veldnamen van Paradox- en SQL-tabellen mogen echter wel spaties en andere tekens bevatten. Als u een dergelijke naam in een dBASE-commando wilt opnemen, plaatst u deze niet-standaard veldnaam tussen dubbele punten.

```
SET FIELDS TO :Oude prijs:     && Veld met spaties tussen dubbele punten plaatsen
```

Plaats een veldnaam eveneens tussen scheidingstekens als u uit een tabel met een onjuiste taalaansturingsidentificatie een dBASE-veld wilt openen waarvan de naam tekens bevat die niet voorkomen in de loaal ingestelde tekenset. DRÜCKER is bijvoorbeeld een geldige veldnaam in het Duits, maar "Ü" is geen geldig teken in de meestal gebruikte tekenset. Zie Appendix C, "Werken met tekensets en taalaansturing" voor meer informatie.

Records toevoegen

Gebruik de commando's APPEND en INSERT als u records aan een tabel wilt toevoegen. In de meeste gevallen gebruikt u APPEND, omdat de records dan automatisch aan de tabel worden toegevoegd (na het laatste record in een tabel met recordnummers). INSERT wordt nauwelijks gebruikt, omdat u de tabel of de bijbehorende index bij elk nieuw record opnieuw moet rangschikken.

Records toevoegen

Met het commando APPEND voegt u een nieuw record toe en geeft u een venster weer waarin u de gegevens kunt bewerken. Met APPEND BLANK voegt u een nieuw record toe en plaatst u de recordaanwijzer in het nieuwe record zonder dat het record wordt weergegeven. Gebruik EDIT om de gegevens van het nieuwe record interactief te bewerken of REPLACE om gegevens aan bepaalde velden toe te voegen.

```
USE BEDRIJF
APPEND BLANK                        && Record toevoegen aan einde van tabel
                                    && BEDRIJF.DBF en recordaanwijzer in dit
                                    && record plaatsen
REPLACE BEDRIJF WITH "VH Klompen"   && Velden bijwerken die zijn toegevoegd
                                    && met APPEND BLANK
```

Records invoegen

De commando's INSERT en INSERT BLANK hebben ongeveer hetzelfde resultaat als APPEND en APPEND BLANK, alleen wordt met INSERT en INSERT BLANK het nieuw record direct na het huidige record ingevoegd.

```
USE BEDRIJF
INSERT BLANK                        && Record invoegen na huidige record
REPLACE BEDRIJF WITH "ACM Papier"   && Ingevoegde record bewerken
```

Gebruik INSERT met de optie BEFORE om een nieuw record in te voegen vóór de huidige positie van de recordaanwijzer.

In Paradox- en SQL-tabellen kunnen APPEND en INSERT soms een andere werking hebben, omdat deze tabellen geen recordnummers kennen. Zie Hoofdstuk 23 voor meer informatie over nieuwe records toevoegen (invoegen).

Nieuwe records toevoegen bij actieve index

Als u een tabel met een index hebt geopend, wordt de werking van zowel APPEND [BLANK] als INSERT [BLANK] hierdoor beïnvloed. In een tabel met een index worden de records gerangschikt in de indexvolgorde en niet op volgorde van recordnummer. Als u een nieuw record toevoegt met APPEND en vervolgens de gegevens ervan bewerkt, wordt het record op de juiste indexpositie in de tabel weergegeven. Als u een nieuw record toevoegt met INSERT en de gegevens ervan bewerkt, wordt het eveneens op de juiste indexpositie weergegeven. (Als de tabel een indexbestand gebruikt, voegt INSERT het nieuwe record eerst toe aan het einde van het bestand.) In beide gevallen wordt de recordaanwijzer na het ingevoegde record geplaatst. Gebruik de optie NOFOLLOW als u wilt dat de recordaanwijzer op het ingevoegde record blijft geplaatst nadat u dit met BROWSE hebt bewerkt. Gebruik REINDEX om alle .NDX- en .MDX-indexbestanden van een tabel bij te werken als u deze bestanden pas hebt geopend nadat u records hebt toegevoegd of bijgewerkt. Als u dBASE-tabellen koppelt met SET RELATION en daarbij de optie INTEGRITY opgeeft, worden sleutelvelden in nieuwe records die aan een subtabel worden toegevoegd ingesteld op waarden die overeenkomen met die in de hoofdtabel. Zie de beschrijving van SET RELATION in *Commando's en functies* voor meer informatie.

Waarden van vorige records kopiëren

Als u het commando SET CARRY samen met APPEND en INSERT gebruikt, kunt u opgeven welke waarden uit het vorige record van de tabel worden gekopieerd.

```
USE BEDRIJF
SET CARRY TO ORDOPBR, TERMKLAS      && Veldwaarden kopiëren in tabel BEDRIJF
APPEND                              && Record toevoegen aan einde van tabel BEDRIJF
```

Records wijzigen

Gebruik het commando REPLACE als u records wilt wijzigen. U kunt elke geldige uitdrukking gebruiken om de waarde van een veld in een of meer records te vervangen. Als u REPLACE gebruikt zonder een aanvullende optie, worden de veldwaarden van het huidige record bijgewerkt. Als u bij REPLACE de opties *<bereik>*, FOR of WHILE opgeeft, worden alle records bijgewerkt die voldoen aan de opgegeven voorwaarden.

```
USE BEDRIJF ORDER BEDRIJF
SEEK "Netbal advies"                && Bepaald record zoeken
IF FOUND()
   REPLACE BEDRIJF WITH Nwe_Naam    && Naam van het veld Bedrijf vervangen
ENDIF
REPLACE ALL COMPCODE WITH Nwe_comp  && Alle veldwaarden van Compcode in alle
                                    && records vervangen
```

Velden vervangen vanuit een array

Gebruik het commando REPLACE FROM ARRAY om de waarden uit velden te vervangen door de waarden die zijn opgeslagen in een array.

```
DECLARE BedrArray[2]                    && Waarden definiëren en toewijzen aan array
BedrArray[1] = x1
BedrArray[2] = x2
USE BEDRIJF ORDER Bedrijf
SEEK xBedrijf                           && Bepaald record zoeken
IF FOUND()
  REPLACE FROM ARRAY CompArray          && Eerste twee velden vervangen door nieuwe waarden
ENDIF
```

In een tweedimensionale array geeft het tweede subscript aan hoeveel kolommen de array bevat, hetgeen overeenkomt met het aantal velden dat u kunt bijwerken. Het eerste subscript geeft aan hoeveel rijen de array bevat, hetgeen overeenkomt met het aantal records dat u kunt bijwerken.

Als u REPLACE FROM ARRAY gebruikt, worden de gegevens uit de eerste rij van de array gekopieerd naar de overeenkomende velden van het huidige record; de gegevens uit de tweede rij worden naar het tweede record gekopieerd enzovoort.

Records markeren en verwijderen

Records worden in twee stappen uit een dBASE-tabel verwijderd:

1 Markeer het record dat u wilt verwijderen met het commando DELETE.

2 Verwijder de gemarkeerde records definitief met PACK.

```
USE AFNEMERS
DELETE FOR OPENBALANS < 750          && Records markeren voor verwijdering die
                                     && zijn geselecteerd met clausule FOR
PACK                                 && Voor verwijdering gemarkeerde records
                                     && definitief verwijderen
```

In Paradox- en SQL-tabellen worden records wél definitief verwijderd met DELETE. U hoeft de records niet eerst te markeren en deze vervolgens definitief te verwijderen met PACK. U kunt de verwijdering bovendien niet ongedaan maken. Als u er zeker van wilt zijn dat een wijziging (zoals een verwijdering) pas wordt doorgevoerd in de tabel als de bewerking succesvol is verlopen, kunt u gebruik maken van transacties.

De functie DELETED() geeft aan of het huidige record voor verwijdering is gemarkeerd.

```
USE AFNEMERS
DELETE FOR OPENBALANS < 750          && Records markeren voor verwijdering die
                                     && zijn geselecteerd met clausule FOR
GOTO 23
? DELETED()                          && True als resulaat geven voor records
                                     && die voor verwijdering zijn gemarkeerd
```

Met het commando RECALL herroept u records, ofwel maakt u de markering van de te verwijderen records ongedaan (als u deze nog niet definitief hebt verwijderd met PACK).

```
RECALL ALL                           && Gemarkeerde records herroepen
```

U kunt bij RECALL, evenals bij DELETE, de opties <*bereik*>, FOR en WHILE opgeven.

Als u dBASE-tabellen wilt kopppelen met SET RELATION, kunt u met de optie INTEGRITY voorkomen dat records in de hoofdtabel worden verwijderd als een subtabel nog gekoppelde records bevat. Zie de beschrijving van SET RELATION in *Commando's en functies* voor meer informatie.

Gemarkeerde records negeren

Met SET DELETED ON worden de records die voor verwijdering zijn gemarkeerd, uitgesloten bij de verwerking van alle volgende commando's. Als SET DELETED is ingesteld op OFF, worden de gemarkeerde records niet uitgesloten.

```
USE AFNEMERS
SET DELETED ON
```

U kunt ook het commando ZAP gebruiken om alle records uit een tabel te verwijderen. Wees voorzichtig als u dit commando gebruikt, omdat alle records dan direct uit de tabel worden verwijderd zonder dat u dit ongedaan kunt maken.

Werken met AUTOMEM-variabelen

Het gebruik van Formulierontwerp en de nieuwe methoden voor het maken van gegevensinvoer- en weergaveformulieren in de Windows-omgeving, heeft de voorkeur boven het gebruik van de traditionele programmeringsmethoden in dBASE, waarbij u de commando's @...SAY...GET gebruikt in combinatie met APPEND en REPLACE.

Als u formulieren ontwerpt waarin gebruikers gegevens kunnen weergeven en bijwerken, kunt u invoervakken voor het formulier definiëren en kenmerken opgeven, zoals DataSource en DataLink, waarmee de invoervakken worden gekoppeld aan de velden in een tabel. U kunt ook kenmerken opgeven waarmee u de navigatie, de verwerking van toetsenbord- en muisacties en andere bewerkingen op het formulier kunt besturen.

Mogelijk beschikt u over dBASE III PLUS- of dBASE IV-programma's die u wilt behouden en waarmee u wilt blijven werken. dBASE voor Windows ondersteunt daarom een nieuw type geheugenvariabele, *AUTOMEM*-variabelen, waarmee u geheugenvariabelen maakt die overeenkomen met de velden in de tabel.

AUTOMEM-variabelen maken

U maakt AUTOMEM-variabelen met USE...AUTOMEM, CLEAR AUTOMEM of STORE AUTOMEM. U kunt de waarden van AUTOMEM-variabelen op dezelfde manier bijwerken als elke andere geheugenvariabele in dBASE. Als u de gegevens in de overeenkomende tabel wilt bijwerken, kunt u APPEND AUTOMEM, INSERT AUTOMEM of REPLACE AUTOMEM opgeven.

```
USE AFNEMERS AUTOMEM          && Lege AUTOMEM-variabelen maken bij openen tabel
CLEAR AUTOMEM                 && Lege AUTOMEM-variabelen maken die overeenkomen met
                              && velden in tabel AFNEMERS.DBF. Geheugenvariabelen
                              && verwijderen als deze al bestaan
STORE AUTOMEM                 && Waarden uit huidige record opslaan in AUTOMEM-
                              && variabelen. Maken indien niet aanwezig
:
REPLACE AUTOMEM               && Huidige records bijwerken met waarden in
                              && AUTOMEM-variabelen
```

REPLACE AUTOMEM werkt alleen het huidige record bij. De opties *<bereik>*, FOR of WHILE, waarmee u alle records kunt bijwerken die aan een bepaalde voorwaarde voldoen, zijn niet beschikbaar voor dit commando.

Met het commando APPEND AUTOMEM kunt u een nieuw record toevoegen, waarna de waarden van de velden in de tabel worden vervangen door de inhoud van de overeenkomende AUTOMEM-variabelen. APPEND AUTOMEM voert dezelfde bewerking uit als APPEND BLANK gevolgd door REPLACE AUTOMEM. Met APPEND AUTOMEM kunt u bovendien gegevens in memovelden plaatsen. U kunt AUTOMEM-variabelen vrijmaken uit het geheugen met het commando RELEASE AUTOMEM.

Werken met memo-, binaire en OLE-velden

In vorige versies van dBASE werden alleen memovelden gebruikt om omvangrijke gegevenselementen, zoals grote stukken tekst, binaire gegevens, afbeeldingen en geluidsgegevens, op te slaan. In dBASE voor Windows zijn hieraan twee nieuwe gegevenstypen toegevoegd: binaire velden en OLE-velden. Hiermee worden speciale opslagmethoden gedefinieerd en ondersteuning van andere Windows-toepassingen waaraan gegevens kunnen worden onttrokken.

Gegevens plaatsen in memovelden

dBASE voor Windows ondersteunt dezelfde bewerkingen voor memovelden als de vorige versies. Gebruik de commando's APPEND MEMO of REPLACE MEMO...FROM om een memoveld in het huidige record van een tabel bij te werken met de inhoud van een bepaald bestand. Gebruik REPLACE MEMO...WITH om een memoveld bij te werken met de inhoud van een array-element.

```
USE KLANTEN
GOTO 1
APPEND MEMO NOTITIES FROM TEKST.TXT        && Inhoud van TEKST.TXT toevoegen aan memo
DECLARE TempMemo[25]                       && Array met elementen maken voor elke regel
                                           && (maximaal 25) voor plaatsing in memoveld
:
GOTO 10
REPLACE MEMO NOTITIES WITH ARRAY TempMemo  && Memoveld bijwerken. Elk array-element
                                           && voegt een regel toe aan het memoveld
? NOOT                                     && Inhoud van memoveld weergeven
```

Als u gegevens uit memovelden weergeeft, bestuurt u het aantal tekens per regel met SET MEMOWIDTH. Met de functie MLINE() kunt u een bepaalde regel uit een memoveld ophalen. U kunt dan zowel het aantal regels opgeven als de breedte van elke regel die wordt opgehaald uit het memoveld.

Memovelden gebruiken in uitdrukkingen

U kunt een memoveld op elke positie in dBASE gebruiken waar een tekenveld of een uitdrukking is toegestaan (u kunt echter geen index maken voor een memo-, binair of OLE-veld). U kunt bijvoorbeeld met de functie AT() zoeken naar een bepaalde tekenreeks in een memoveld of SUBSTR() gebruiken om een subreeks uit een memoveld op te halen. U kunt ook gebruik maken van operatoren, zoals plus (+) of min (-), om de inhoud van een memoveld en een andere tekenreeks bij elkaar op te tellen of van elkaar af te trekken als u een uitdrukking definieert.

```
USE Klanten
? SUBSTR(Notities, 1, 80)                    && De eerste 80 tekens van het memoveld
                                             && Notities in het huidige record ophalen
? SUBST(Notities, AT("nabestelling",Notities),80) && Tekenreeks uit het memoveld Notities
                                             && als resultaat geven als de zoekreeks
                                             && "nabestelling" wordt gevonden
? "Inhoud van memo" + Notities              && Tekenreeks toevoegen aan memoveld Notities
```

RAT(), STUFF(), TRANSFORM() en UPPER zijn andere functies waarmee u reeksen kunt bewerken.

Gegevens plaatsen in binaire en OLE-velden

Met de commando's REPLACE BINARY en REPLACE OLE kunt u binaire velden en OLE-velden bijwerken vanuit respectievelijk binaire bestanden (bijvoorbeeld .BMP, .PCX en .WAV) of OLE-documenten.

```
REPLACE BINARY PICTFLD FROM ZEBRA.BMP       && Inhoud van ZEBRA.BMP opslaan in
                                            && binair veld
REPLACE OLE WORDDOC FROM BRIEF.DOC          && OLE-veld koppelen aan document dat is
                                            && opgeslagen door andere Windows-toepassing
```

REPLACE BINARY kent de optie TYPE, waarmee u een binair bestand kunt vervangen dat door de gebruiker is gedefinieerd.

Gegevens weergeven in binaire en OLE-velden

dBASE voor Windows verschaft twee nieuwe commando's waarmee u afbeeldings- en geluidsgegevens bewerkt in binaire gegevensvelden. U gebruikt RESTORE IMAGE als u de inhoud wilt weergeven van een binair veld in het huidige record van de tabel dat afbeeldingen bevat (binaire gegevens uit .BMP- en .PCX-bestanden). Als u dit commando opgeeft, wordt de afbeelding die is opgeslagen in het opgegeven binaire veld van het huidige record, weergegeven in een venster. Op soortgelijke wijze gebruikt u het commando PLAY SOUND om de geluidsgegevens (.WAV) uit een binair veld af te spelen.

U kunt BROWSE of EDIT gebruiken om een afbeelding of geluid afkomstig uit een binair veld, weer te geven of af te spelen door te dubbelklikken op het veld. U geeft afbeeldingen in een formulier weer door een afbeeldingsobject te definiëren en het kenmerkDataSource te gebruiken om dit te koppelen aan een binair veld in de tabel. Bijvoorbeeld:

```
CLASS NwAfbld OF FORM
  :
DEFINE IMAGE Afbld1 OF THIS;
  PROPERTY;
    DataSource "BINARY Dieren->BMP", ;
      :
ENDCLASS
```

U definieert het standaardformaat en de standaardlokatie van het venster waarin de afbeelding wordt weergegeven met de kenmerken die u aan het afbeeldingsobject toewijst.

Als u een OLE-document wilt openen in een formulier, definieert u een OLE-object en gebruikt u het kenmerk DataLink om het object te koppelen aan een OLE-veld in een tabel.

Gegevens uit binaire velden kopiëren

U gebruikt COPY BINARY om de inhoud van een binair veld naar een bestand te kopiëren.

```
COPY BINARY AFBEELDING TO UIT.PCX        && Inhoud van binaire veld AFBEELDING
                                         && kopiëren naar bestand UIT.PCX
```

Hoofdstuk

19

Samenhang en volgorde van records

dBASE beschikt over hulpmiddelen waarmee u gegevens kunt ordenen en weergeven, zodat u gemakkelijk zinnige informatie kunt onttrekken aan de ruwe gegevens in de tabel. Bij hulpmiddelen zoals query's, formulieren en rapporten worden indexen gebruikt om tabellen te doorzoeken en deelverzamelingen van dezelfde gegevens op allerlei manieren weer te geven. De volgorde waarin records in een tabel worden weergegeven, is afhankelijk van de hoofdindex waarmee u de rangschikking bestuurt. U kunt de volgorde van records dus wijzigen.

In dit hoofdstuk worden de onderwerpen en methoden behandeld die betrekking hebben op het gebruik van indexen in de programmering onder dBASE. Zie Hoofdstuk 2 in het *Handboek* voor een beschrijving van het basisprincipe van indexeren.

In dit hoofdstuk worden de volgende onderwerpen behandeld:

- Indexbestanden en indexlabels gebruiken
- Indexlabels maken en verwijderen
- Relaties instellen en gegevensintegriteit onderhouden

Indexeren en sorteren

Records sorteren is een dynamische procedure die onafhankelijk is van de werkelijke volgorde waarop deze in de tabel verschijnen. De werkelijke volgorde van de records in de tabel is hierbij zelfs niet van belang. Deze volgorde blijft zelfs bij een perfect gesorteerde tabel niet intact, omdat de gegevens altijd worden gewijzigd.

Een index maakt een gesorteerde verwijzing naar een tabel op basis van de sleutelvelden die u selecteert. Als u een index gebruikt, worden de gegevens gesorteerd naar de geselecteerde indexsleutel weergegeven. De werkelijke volgorde van de tabel blijft ongewijzigd en er wordt evenmin een kopie van de tabel gemaakt.

Meestal kunt u beter gebruik maken van indexen dan van het commando SORT. Met dit commando wordt namelijk de werkelijke volgorde van de gegevens gewijzigd op basis van het sorteerveld dat u selecteert. Vervolgens worden de gegevens naar een nieuwe tabel gekopieerd.

Opmerking Zowel bij indexeren als sorteren wordt de volgorde waarin tekenvelden met speciale tekens (tekens met accenten) worden gerangschikt, beïnvloed door de ingestelde taalaansturing. Zie Appendix C voor meer informatie.

Records sorteren

Een beginnend gebruiker van een databasetoepassing vindt het misschien vanzelfsprekend dat de gegevens worden gesorteerd, omdat bekende statische databases, zoals woordenboeken, telefoonboeken en naslagwerken, meestal op alfabetische volgorde worden gesorteerd en gedrukt. Een tabel maakt echter gebruik van logische indexen om gegevens in dynamische tabellen te sorteren en te zoeken, waardoor de werkelijke volgorde van de records in de tabel niet meer van belang is. Daarom kunt u het commando SORT normaal gesproken beter niet gebruiken.

U kunt beperkt gebruik maken van SORT, bijvoorbeeld als u een nieuwe tabel maakt en verspreidt en daarna de gegevens voorlopig niet meer wijzigt. Van de gegevens van een seizoencatalogus met artikelen, weet u bijvoorbeeld dat deze de komende maanden niet meer worden gewijzigd of verwijderd. Zie *Commando's en functies* voor meer informatie over het commando SORT.

Tabellen samenvoegen

Met het commando JOIN voegt u tabellen samen, zodat er een nieuwe tabel ontstaat waarvan de velden in een speciale volgorde zijn gesorteerd. U kunt het volgende doen met JOIN:

- SORT en JOIN gebruiken om een tijdelijke tabel voor een rapport te maken, waarbij een deelverzameling van de gegevens wordt uitgelicht en afzonderlijk gesorteerd in een nieuwe tabel.
- Een genormaliseerde opzoektabel in een hoofdtabel opnemen, vooral wanneer de opzoektabel statische gegevens bevat.
- Nauw verwante velden naast elkaar in de tabel plaatsen, zodat u eenvoudiger kunt bladeren en sneller kunt indexeren.

Opmerking Het commando JOIN is niet bijzonder geschikt om nieuwe tabellen te maken. De tabellen kunnen dan erg groot worden en dubbele velden bevatten.

U kunt ook nieuwe tabellen maken in Query-ontwerp door velden uit verschillende tabellen te combineren. U kiest dan **Query | Resultaat naar nieuwe tabel** om het resultaat op te slaan. Selecteer de velden die u wilt combineren zorgvuldig, zodat gegevens niet dubbel worden opegnomen. Zie Hoofdstuk 1 in het *Handboek* voor meer informatie over de basisprincipes van tabelontwerp.

Indexlabels en indexbestanden gebruiken

Met indexlabels wijzigt u de volgorde waarin de tabel wordt doorzocht. De records worden gesorteerd in de volgorde waarin deze in het indexbestand verschijnen. Hiermee kunt u snel de weergave van gegevens wijzigen en naar bepaalde records zoeken.

Er zijn twee typen indexbestanden: meervoudige indexbestanden (.MDX) en enkelvoudige indexbestanden (.NDX). Enkelvoudige indexbestanden zijn te vergelijken met één enkel indexlabel in een .MDX-bestand. Een enkelvoudig indexbestand wordt voornamelijk gebruikt vanwege de compatibiliteit met eerdere versies van dBASE. U kunt een enkelvoudig indexbestand echter ook incidenteel gebruiken voor een tijdelijke, eenmalige index. U opent een enkelvoudig indexbestand als volgt:

1 Gebruik het commando USE om bijvoorbeeld de tabel KLANTEN.DBF vanuit Navigator te openen.

2 Maak een index voor bijvoorbeeld het veld Regio. Typ het volgende in het commandovenster:

```
INDEX ON REGIO TO REGIO.NDX
```

De tabel Klanten wordt opnieuw gesorteerd op basis van Regio. U kunt deze index bijvoorbeeld gebruiken om alle klanten in een bepaalde regio weer te geven, zodat u deze kunt bellen als het tarief gunstig is. Verwijder het .NDX-bestand als u het niet meer nodig hebt.

Als u een tabel maakt, wordt automatisch een bijbehorend .MDX-bestand met een produktie-index gemaakt. Dit .MDX-bestand wordt automatisch bijgewerkt zodra u de gegevens in de bijbehorende tabel wijzigt.

.NDX-bestanden converteren naar .MDX-bestanden

Tenzij compatibiliteit is vereist met eerdere versies van dBASE, kunt u enkelvoudige indexbestanden beter converteren naar labels in het produktie-MDX-bestand. U kopieert enkelvoudige indexbestanden als indexlabels naar het produktie-MDX-bestand. Het gebruik van meervoudige indexbestanden biedt de volgende voordelen:

- Produktie-MDX-bestanden worden automatisch bijgewerkt, zodat tabellen en indexlabels op elkaar blijven afgestemd.
- .MDX-bestanden worden sneller uitgevoerd dan enkelvoudige indexbestanden.
- De labels in een geopend .MDX-bestand zijn altijd beschikbaar.
- Elk bestand kan maximaal 47 labels bevatten, zodat u minder bestanden hoeft te openen. Dit komt met name de uitvoeringssnelheid van grote tabellen en indexen ten goede.

Als u .NDX-bestanden wilt converteren naar .MDX-labels, kopieert u deze naar een .MDX-bestand. Als u geen .MDX-bestand als doelbestand opgeeft, wordt het indexlabel naar het geopende produktie-MDX-bestand gekopieerd. Als u meerdere .NDX-bestanden wilt opslaan als labels, kunt u dit alleen doen als deze alle zijn geopend.

Als u bijvoorbeeld twee .NDX-bestanden wilt opslaan als labels in het produktie-MDX-bestand van KLANTEN.DBF, kunt u de volgende syntaxis invoeren:

```
USE KLANTEN EXCLUSIVE INDEX REGIO.NDX, NAAM.NDX      && Gewenste bestanden openen

COPY INDEXES REGIO.NDX, NAAM.NDX                     && Indexbestanden als labels kopiëren
                                                     && naar produktie-MDX-bestand

USE                                                  && Alle bestanden inclusief
                                                     && produktie-MDX sluiten en
                                                     && indexen bijwerken
```

.MDX-labels converteren naar .NDX-bestanden

U kunt ook een enkelvoudig indexbestand maken van een indexlabel in een .MDX-bestand. U kunt bijvoorbeeld met een .NDX-bestand werken als u de index zowel met dBASE III PLUS als dBASE voor Windows wilt gebruiken. Bijvoorbeeld:

```
USE KLANTEN                          && Tabel en produktie-MDX-bestand openen
COPY TAG Klantnr TO KLANTNR.NDX      && NDX-bestand maken van label
SET INDEX TO KLANTNR.NDX             && Nieuwe enkelvoudige index gebruiken als
                                     && hoofdindex
```

Gebruik het commando SEEK als u de records wilt zoeken die Klantnr als hoofdindex hebben. Zie Hoofdstuk 20 voor meer informatie over records zoeken.

Indexen onderhouden

U moet een index regelmatig onderhouden. Dit geldt vooral voor niet-produktie-indexen, omdat u deze wellicht vergeet te openen als een tabel een paar honderd indexlabels bevat. Bovendien is het mogelijk dat een .MDX-bestand labels bevat die u niet langer nodig hebt. U kunt de verouderde labels dan verwijderen uit het produktie-MDX-bestand en de overige .MDX-bestanden. Verwijder ook elk .NDX-bestand dat niet meer wordt gebruikt.

U verwijdert een een overbodig .NDX-bestand vanuit het commandovenster. Het indexbestand Regio verwijdert u bijvoorbeeld met het volgende commando:

```
DELETE FILE REGIO.NDX
```

Als u overbodige labels verwijdert, wordt de index sneller doorzocht. Als het .MDX-bestand in het huidige werkgebied is geopend, hoeft u de naam ervan niet op te geven. Als u bijvoorbeeld geen controle meer wilt uitoefenen op de verzenddatum, verwijdert u het label in het commandovenster:

```
DELETE TAG VERZENDDAT OF ORDERS.MDX
```

Geef de optie NOSAVE op bij SET ORDER TO als u een indexlabel gebruikt voor een eenmalige zoekopdracht. Met NOSAVE maakt u een tijdelijk label in het produktie-MDX-bestand of in het .MDX- of .NDX-bestand dat u opgeeft. Als u de index sluit, wordt het label verwijderd. Bijvoorbeeld:

```
SET ORDER TO VERZENDDAT OF ORDERS.MDX NOSAVE
```

Nadat u de index hebt gebruikt, sluit u de hoofd-NDX- of de niet-produktie-MDX-index met het volgende commando:

```
SET ORDER TO
```

Het label VERZENDDAT wordt verwijderd uit het .MDX-bestand of het .NDX-bestand.

Als u de optie NOSAVE gebruikt met het produktie-MDX-bestand, sluit u de tabel met CLOSE ALL of typt u het commando USE zonder bestandsnaam.

Zorg dat u geen belangrijke labels verwijderd met deze optie. Het gebruik van NOSAVE wordt dan wellicht tijdrovend omdat u de labels opnieuw moet maken.

Als u gegevens vaak wijzigt zonder opnieuw te indexeren, bestaat de kans dat de indexlabels niet meer op de gegevens zijn afgestemd. Zorg daarom dat u omvangrijke, veelgebruikte tabellen regelmatig opnieuw indexeert. U gaat hiervoor als volgt te werk:

1 Open elke tabel in een afzonderlijk werkgebied.

2 Open de produktie-MDX-bestanden en alle overige bestanden die u wilt bijwerken. Als u indexcommando's wilt gebruiken in een mult-user-omgeving, opent u tabellen met EXCLUSIVE .

3 Gebruik het commando REINDEX om alle indexlabels in de geopende bestanden bij te werken. Bijvoorbeeld:

```
USE KLANTEN IN 1 INDEX KLANT.MDX EXCLUSIVE     && Tabel exclusief openen en
                                                && produktie-MDX-bestand met een
                                                && aanvullend MDX-bestand openen
USE ORDERS IN 2 INDEX KLANTNR                   && Tweede tabel met een aanvullend
                                                && MDX-bestand openen
REINDEX                                         && Elk indexlabel in geopende bestanden
                                                && bijwerken
```

Indexlabels maken

U kunt een indexsleutel maken voor een indexlabel op basis van een veld of een uitdrukking waarin één of meer veldnamen uit een dBASE-tabel zijn opgenomen. U kunt geen index maken op basis van memo-, binaire of OLE-velden.

Als u meer dan één veldnaam gebruikt, voegt u twee tekenreeksen samen met de aaneenschakelingsoperator plus (+) of verplaatst u de volgspaties naar het einde van de laatste reeks met de minoperator (-). Als de tabel numerieke sleutels bevat, kunt u een indexuitdrukking opgeven met de rekenkundige operatoren (-, *, /), waarmee u respectievelijk kunt aftrekken, vermenigvuldigen en delen. De indexuitdrukking moet voor elk record een indexsleutel van gelijke lengte opleveren.

U kunt in dBASE indexen gebruiken afkomstig uit Paradox- en SQL-tabellen. Bovendien kunt u indexen maken op basis van Paradox- en SQL-tabellen. Zie Hoofdstuk 23 voor meer informatie over het gebruik van niet-dBASE-indexen.

Datumgegevens converteren

Als u datumgegevens wilt combineren met andere gegevenstypen in een indexsleutel, converteert u de datumgegevens naar tekens. De datumopmaak wordt dan eerst

gewijzigd in JMD met eeuwaanduiding. In de volgende conversie worden de cijfers van de datum van links naar rechts weergegeven in volgorde van belangrijkheid. Bijvoorbeeld:

```
Chardate = DTOS(BETDATUM)
? Chardate              && 19940327 als resultaat geven
```

Als een tabel bijvoorbeeld onder andere de velden ACHTERNAAM, VOORNAAM en BETDATUM bevat, gebruikt u de functie DTOS() om datum- en naamgegevens aaneen te schakelen om te zien welke klanten op of voor de vervaldatum hebben betaald. Bijvoorbeeld:

```
INDEX ON ACHTERNAAM+VOORNAAM+DTOS(BETDATUM) TAG Betaling OF KLANT.MDX;
   FOR DTOS(BETDATUM) <= 19940301
```

Logische gegevens converteren
Als u in een indexsleutel logische velden wilt gebruiken in combinatie met andere gegevenstypen, converteert u deze naar tekens. Het is niet mogelijk om logische gegevens direct met een functie te converteren naar tekens, maar wel indirect met de functie IIF(). Stel dat de tabel KLANTEN het logische veld Gebeld bevat. U kunt dan een index maken waarmee u de klanten weergeeft die u niet hebt gebeld:

```
USE KLANTEN
INDEX ON IIF(GEBELD, "J", "N" ) TAG Bellen          && Records met nee eerst rangschikken,
                                                    && daarna met ja
```

U hebt nu het indexlabel Bellen gemaakt, waarmee u eerst de klanten weergeeft die nog niet zijn gebeld. Als u dezelfde informatie snel wilt opzoeken, kunt u ook een query gebruiken. Als u deze informatie regelmatig nodig hebt, kunt u wellicht beter een index gebruiken.

Dubbele sleutels besturen

SET UNIQUE ON of INDEX ON <*sleuteluitdrukking*> UNIQUE hebben dezelfde functie: beide verwerken alleen het eerste record als meerdere records dezelfde sleutelwaarde hebben. Als u een dubbel record wijzigt terwijl de hoofdindex met UNIQUE is geopend, wordt alleen de eerste instantie van dat record gewijzigd of bijgewerkt. Het indexlabel dat u hebt gemaakt met UNIQUE blijft bovendien UNIQUE als u dit opnieuw indexeert, ongeacht of u hierbij SET UNIQUE hebt ingesteld op OFF of ON.

Opmerking UNIQUE verbergt de dubbele records in een tabel. U kunt dubbele records niet weergeven en bijwerken.

UNIQUE zoekt de eerste instantie dat een dubbel veld voorkomt en sluit daarmee dubbele velden uit. Bij sommige velden, zoals naamvelden, is het soms onvermijdelijk dat deze dubbele records bevatten. Gebruik de methoden voor gegevenscontrole om gebruikersinvoer in tabellen te accepteren.

Met de functie KEYMATCH() controleert u of een bepaalde uitdrukking reeds aanwezig is in een index. Gebruik deze functie als u records toevoegt aan een tabel. U voorkomt dan dat er dubbele records worden toegevoegd.

Indexen openen en sluiten

U kunt elk indexbestand, behalve het produktie-MDX-bestand, desgewenst openen en sluiten. Als u een .MDX-bestand opent, kunt u bovendien het label opgeven dat u als hoofdindex wilt gebruiken.

U kunt op hetzelfde moment een .MDX-index openen en een label activeren. Bijvoorbeeld:

```
USE KLANTEN INDEX NAMEN.MDX ORDER TAG AchterNaam OF NAMEN.MDX
```

U kunt het actieve label vervangen door een ander label uit hetzelfde of een ander meervoudig indexbestand. Bijvoorbeeld:

```
SET ORDER TO TAG VoorNaam
```

U kunt verschillende indexbestanden tegelijk openen. Bijvoorbeeld:

```
USE KLANTEN INDEX NAMEN.MDX, KLANT.MDX, GELUID.NDX
```

U kunt verschillende indexbestanden tegelijk openen en een label als hoofdindex instellen. Bijvoorbeeld:

```
SET INDEX TO NAMEN.MDX, KLANT.MDX, GELUID.NDX ORDER AchterNaam OF KLANT
```

U sluit een index op een van de volgende manieren:

- Met SET INDEX TO of SET ORDER TO zonder argument sluit u de hoofdindex, behalve als dit het produktie-MDX-bestand is.
- Met CLOSE INDEX sluit u alle geopende indexbestanden, behalve het produktie-MDX-bestand. Sluit de tabel als u het produktie-MDX-bestand wilt sluiten.
- Met USE zonder argument of CLOSE TABLES sluit u alle geopende tabellen en bijbehorende indexen.

Volgorde van records besturen

Met het commando SET KEY TO RANGE sluit u het bovenste en onderste bereik van de records uit van weergave met behulp van de bestaande indexlabels. Deze methode werkt sneller dan voorwaardelijke indexering met FOR. Het bereik dat u instelt met SET KEY wordt direct van kracht en heeft voorrang boven het commando SET FILTER. Als u echter een indexsleutel gebruikt voor SET FILTER, zijn beide methoden even snel.

Als u zowel SET KEY als SET FILTER gebruikt, kunt u alleen de records weergeven of wijzigen die aan beide selectiecriteria voldoen.

U kunt SET KEY TO RANGE en SET FILTER TO in dBASE-tabellen en in niet-dBASE-tabellen gebruiken. Met SET FILTER kunt u flexibele selectiecriteria hanteren. Zie Hoofdstuk 23 "Werken met niet-dBASE-tabellen" voor meer informatie.

Indexstatus bepalen

Met de ondersteuningsfuncties voor indexen kunt u de naam van het hoofdindexbestand of logische gegevens over geopende indexlabels en -bestanden

opvragen. U kunt deze informatie in programma's gebruiken om sprongopdrachten uit te voeren, hoofdindexen te wijzigen, gegevenscontrole uit te voeren enzovoort. In Tabel 19.1 ziet u een overzicht van deze commando's en functies. Zie *Commando's en functies* voor een overzicht van de parameters die u bij deze functies kunt gebruiken..

Tabel 19.1 Indexstatus bepalen

Indexfuncties	Resultaat
DESCENDING()	(.T.) als de hoofdindex aflopend is. (.F.) als de hoofdindex niet-aflopend is of als het een indextype betreft dat geen aflopende indexen ondersteunt.
FOR()	De voorwaarde FOR van de hoofdindex of een nulreeks als er een .NDX-index is geopend.
KEY()	Huidige hoofdindexsleutel.
KEYMATCH()	(.T.) als een uitdrukking wordt gevonden in de opgegeven index. (.F.) als geen uitdrukking wordt gevonden in de opgegeven index.
MDX()	De naam van het hoofd-MDX-bestand of een nulreeks als hoofdindex geen .MDX-bestand is.
NDX()	De naam van het hoofd-NDX-bestand of een nulreeks als hoofdindex geen .MDX-bestand is.
ORDER()	Resulteert in de naam van de hoofdindex in het huidige werkgebied.
TAG()	De naam van het hoofdindexlabel.
TAGCOUNT()	Totaal aantal indexen in het huidige werkgebied.
TAGNO()	Positie van de hoofdindex in het .MDX-bestand.
UNIQUE()	(.T.) als de hoofdindex is ingesteld op UNIQUE. (.F.) als dit niet het geval is.

In de volgende voorbeelden wordt getoond hoe u indexfuncties gebruikt:

```
USE KLANTEN EXCLUSIVE
INDEX ON KLANTNR TAG Nr
INDEX ON NAAM TAG BEDR
? MDX()                          && Klanten.mdx als resultaat geven
? TAGCOUNT()                     && 2 als resultaat geven
? TAGNO()                        && 2 als resultaat geven
? ORDER()                        && BEDR als resultaat geven
SET ORDER TO Nr                  && Hoofdindex wijzigen
? ORDER()                        && Nr als resultaat geven
? TAG()                          && Nr als resultaat geven
```

Indexeringsmethoden

De gegevens in een dBASE-tabel worden regelmatig gewijzigd. In de meeste database-applicaties kunt u de tabellen bijwerken met programma's waarin dBASE-commando's worden gebruikt, zoals BLANK, REPLACE, REPLACE FROM ARRAY of UPDATE.

REINDEX gebruiken

De meeste applicaties bevatten te veel gegevens om handmatig bij te werken. Bovendien worden veel records wellicht een aantal keren per dag gewijzigd. Als bijvoorbeeld door administratief personeel of met geautomatiseerde kassa-apparatuur gegevens in tabellen zijn ingevoerd, worden commando's zoals BLANK, UPDATE, APPEND of REPLACE gebruikt om deze gegevens bij te werken in grotere tabellen.

Deze commando's kennen alle REINDEX als optioneel sleutelwoord. Als u REINDEX gebruikt met deze commando's, wordt de wijziging van records beter uitgevoerd. Met deze optie zorgt u dat een index pas wordt bijgewerkt nadat u alle records hebt gewijzigd. Als u REINDEX niet gebruikt en u vervangt een sleutelveld in de hoofdindex, kan de recordaanwijzer worden verplaatst als hierdoor de positie van een record in de index wordt gewijzigd. Als de recordaanwijzer wordt verplaatst, gaat het overzicht op de globale vervanging verloren wat tot onvoorspelbare resultaten kan leiden.

SET KEY TO gebruiken

Met het commando SET KEY TO RANGE geeft u de ondergrens en de bovengrens aan van de records die u wilt verwerken, of zoekt u de exacte overeenkomst als u één waarde gebruikt voor beide grenzen. Met dit commando kunt u bestaande indexlabels gebruiken en hoeft u geen nieuwe indexen te maken. Met SET KEY kunt u records op dezelfde manier uitsluiten als met INDEX ON <*sleutel*> FOR <*voorwaarde*>.

Als u de records met te betalen bedragen tussen één en tweeduizend gulden wilt bijwerken, zoekt u met de volgende code alleen de records die aan het sleutelbereik voldoen:

```
USE ORDERS
SET ORDER TO TAG BetBedrag      && Label in geopende index voor dit veld
SET KEY TO RANGE 1.00, 2000     && Onder- en bovengrens voor sleutelveld BetBedrag beperken
BROWSE                          && Alleen records weergeven die aan bereik voldoen
GO TOP
```

Voorwaardelijke indexering met FOR werkt langzamer omdat u bestaande indexlabels bijwerkt om records te selecteren die aan bepaalde voorwaarden voldoen. Dit kost tijd omdat er dan veel records worden geëvalueerd om een deelverzameling te maken. U kunt de volgorde van de gegevens daarom beter wijzigen met het commando SET KEY TO RANGE of met de filtervoorwaarden die u invoert in Query-ontwerp.

Relatie tussen tabellen instellen

In Query-ontwerp kunt u een query maken ofwel een *weergave* van gegevens maken op basis van de gegevens in een of meer onderliggende tabellen. U kunt het volgende doen in Query-ontwerp:

- Tabellen openen
- Velden selecteren om weer te geven
- Velden selecteren waarop records worden gesorteerd

- Relatie instellen voor sleutelvelden

Zie het *Handboek* voor meer informatie over het gebruik van Query-ontwerp.

In Query-ontwerp wordt code (opdrachten in dBASE-taal waarmee de query wordt gedefinieerd) gegenereerd als u de query opslaat naar een bestand. Met de Programma-editor kunt u bestanden openen die u in Query-ontwerp hebt gemaakt. Vervolgens kunt u de gegenereerde code opnieuw in uw applicaties gebruiken.

U kunt de code waarmee u een relatie tussen tabellen instelt ook direct schrijven zonder Query-ontwerp te gebruiken. Wellicht vindt u het gemakkelijker om eerst Query-ontwerp te gebruiken en daarna de gegenereerde code te wijzigen.

SET RELATION gebruiken

Met het commando SET RELATION definieert u de koppeling tussen de tabellen in een query. U legt een relatie tussen geopende tabellen op basis van een gemeenschappelijk sleutelveld of een gemeenschappelijke uitdrukking. Als u een relatie hebt ingesteld, wordt de tabel in het huidige werkgebied de *hoofd*tabel genoemd en wordt een tabel die door de opgegeven sleutel is gekoppeld aan de hoofdtabel, een *sub*tabel genoemd. U kunt meerdere relaties tegelijk definiëren op basis van één *hoofd*tabel met de optie ADDITIVE of door meerdere relaties op te geven bij het commando SET RELATION.

Voordat u een relatie instelt, opent u elke tabel in een afzonderlijk werkgebied. Verder maakt u een index voor de subtabel op basis van het sleutelveld of de uitdrukking waarop u de tabellen koppelt. De sleutel kan uit één veld bestaan of uit een verzameling aaneengeschakelde velden die zich in elke tabel bevinden. Met SET EXACT ON geeft u aan dat de gekoppelde velden precies met elkaar moeten overeenkomen.

Stel dat u een tabel met informatie over klanten koppelt aan een tweede tabel met bestellingen. Als u de twee tabellen koppelt op basis van het klantnummer, kunt u de naam en het adres van een bepaalde klant weergeven samen met informatie over de bestellingen van die klant.

```
SET EXACT ON
USE KLANTEN.DBF                      && Eerste tabel openen
SELECT 2                             && Tweede werkgebied selecteren
USE ORDERS.DBF ORDER TAG Klantnr     && Tweede tabel openen, gesorteerd op
                                     && label Klantnr
SELECT 1                             && Werkgebied van hoofdtabel actief maken
SET RELATION TO Klantnr INTO ORDERS  && Tabellen koppelen op basis van veld KLANTNR
```

Gegevensintegriteit in gerelateerde tabellen uitvoeren

Met de opties SET RELATION...CONSTRAIN en INTEGRITY geeft u regels op voor gegevensintegriteit en bepaalt u het effect dat het bijwerken en verwijderen van records heeft op hoofd- en subtabellen. De integriteitsregels gelden alleen voor dBASE-tabellen. Paradox- en SQL-tabellen beschikken over een eigen systeem voor integriteitscontrole. In Query-ontwerp worden deze sleutelwoorden automatisch opgenomen in de gegenereerde code en de integriteitsregels dus automatisch uitgevoerd. In dBASE wordt de gegevensintegriteit als volgt gecontroleerd en uitgevoerd:

- Met SET RELATION stelt u het sleutelveld in de subtabel gedurende de relatie in op alleen-lezen. U voorkomt dan dat er subrecords worden gemaakt waarvoor geen overeenkomstige hoofdrecords bestaan.
- Met het sleutelwoord CONSTRAIN van SET RELATION geeft u alleen de records uit de subtabel weer die overeenkomen met de records in de hoofdtabel. U hoeft dan geen filtervoorwaarde in te stellen in de subtabel om overeenkomstige records te zoeken.
- Met het sleutelwoord INTEGRITY van SET RELATION voert u de integriteit uit van overeenkomstige records in hoofd- en subtabel. Stel dat u de integriteitscontrole tussen twee tabellen inschakelt en vervolgens een formulier maakt op basis van deze tabellen. Als u dan een record in de hoofdtabel wilt verwijderen, verschijnt er een dialoogvenster waarin u opgeeft of u de overeenkomstige records in de subtabel eveneens wilt verwijderen (trapsgewijs wijzigen) of de verwijdering van records wilt annuleren. U voorkomt dan dat er *zwevende* records in de subtabel ontstaan. Bij het sleutelwoord INTEGRITY kunt u de opties CASCADE en RESTRICTED gebruiken.
  - Met CASCADE worden subrecords automatisch verwijderd zodra u de overeenkomstige records in de hoofdtabel verwijdert. Deze optie staat trapsgewijze verwijdering vanuit de subtabel naar de hoofdtabel echter niet toe. Gebruik de optie RESTRICTED om te voorkomen dat gerelateerde subrecords worden verwijderd.
  - Met RESTRICTED voorkomt u dat records uit de subtabel worden verwijderd die een overeenkomend record in de hoofdtabel hebben. Er verschijnt dan een foutmelding.

Als u in Query-ontwerp de optie **1 tot n** inschakelt, wordt het sleutelwoord CONSTRAIN toegevoegd aan het commando SET RELATION in de gegenereerde code. In Query-ontwerp wordt dan ook SET SKIP toegevoegd aan de code zodat alle records in de subtabellen worden verwerkt.

In de volgende, door Query-ontwerp gegenereerde code wordt getoond hoe u een relatie legt tussen KLANTEN.DBF en ORDERS.DBF. De code werkt alleen als de tabellen zijn geïndexeerd op het sleutelveld.

```
CLOSE DATABASES                    && Alle overbodige tabellen sluiten
SET EXACT ON                       && Exacte overeenkomst tussen records opgeven
SELECT 1                           && Werkgebied selecteren
USE KLANTEN.DBF                    && Eerste tabel openen
SELECT 2                           && Tweede werkgebied selecteren
USE ORDERS.DBF ORDER TAG Klantnr   && Tweede tabel openen en volgorde instellen op
                                   && basis van label Klantnr
SELECT 1                           && Werkgebied van hoofdtabel activeren
SET RELATION TO Klantnr;
INTO ORDERS CONSTRAIN INTEGRITY    && Relatie instellen op basis van sleutelveld
                                   && KLANTNR en sleutelwoorden gebruiken om
                                   && te zorgen dat records overeenkomen en
                                   && gegevensintegriteit wordt uitgevoerd
SET SKIP TO ORDERS                 && SKIP instellen op subtabel
SET FILTER TO FOUND(2)             && Filtervoorwaarde voor subtabel met
                                   && werkgebiedalias 2
GO TOP                             && Zoeken starten vanaf begin van tabel
```

Records filteren in gerelateerde tabellen

Gebruik SET KEY TO om een bereik vast te leggen. Gebruik SET FILTER TO *<uitdrukking>* als het niet mogelijk is om een bereik vast te leggen.

Als u een filter instelt voor geïndexeerde velden, wordt de uitdrukking met SET FILTER TO *<uitdrukking>* eenmaal geëvalueerd, waarna de filter wordt gemaakt. Het resultaat is dus hetzelfde als met SET KEY TO RANGE.

```
SET FILTER TO FOUND(<subtabelalias>)
```

Als een record in de hoofdtabel geen overeenkomend record in de subtabel heeft, resulteert de functie FOUND() in False, waarna het hoofdrecord uit de weergave wordt weggefilterd. Deze procedure wordt herhaald totdat de recordaanwijzer het einde van het bestand van de subtabel heeft bereikt, zodat EOF() is True voor de subtabelalias.

In Query-ontwerp kunt u interactief filtervoorwaarden maken op basis van velden in de tabel. U geeft de filtervoorwaarden op met behulp van relationele operatoren en uitdrukkingen in de ruimte onder de veldnaam in het bestandsschema. U kunt ook het voorwaardenvak van Query-ontwerp gebruiken.

Records opnemen in gerelateerde tabellen

Als u SET RELATION TO gebruikt, worden standaard alle records in de subtabel weggefilterd, behalve het eerste record dat overeenkomt met het record in de hoofdtabel.

Als u alle records in de subtabel wilt bekijken, selecteert u het werkgebied van de subtabel of gebruikt u SET SKIP TO om alle records weer te geven die overeenkomende records in de hoofdtabel hebben. Bijvoorbeeld:

```
SET SKIP TO <subtabelalias> [, <subtabelalias2>]...
```

U gebruikt twee of meer aliassen als meerdere tabellen in de relatiereeks zijn opgenomen. Als u bij SKIP de werkgebieden van de subtabellen opgeeft, worden alle records in de subtabellen verwerkt die overeenkomende records in de hoofdtabel hebben.

Als er een lijst met subtabellen is, begint de verwerking bij de laatste subtabel in de lijst.

U hoeft niet alle subtabellen op te nemen in het commando SET SKIP TO. U neemt alleen de subtabellen op die een één-op-meer-relatie hebben met de hoofdtabel.

De tabel Regels kan bijvoorbeeld meerdere records bevatten voor elke klant die meer dan één bestelling heeft geplaatst. U gebruikt dan de commando's SET SKIP en SET FILTER om de regels voor elke order van elke klant te verwerken. Bijvoorbeeld:

```
CLOSE DATABASES
SET EXACT ON
SELECT 1
USE ORDERS.DBF
SELECT 2
USE KLANTEN.DBF ORDER TAG Klantnr
SELECT 3
USE REGELS.DBF ORDER TAG Ordernr
SELECT 1
SET RELATION TO Klantnr INTO KLANTEN, Ordernr INTO REGELS
SET SKIP TO KLANTEN
SET FILTER TO FOUND(2) .AND. FOUND(3)      && Filtervoorwaarden instellen in subtabellen
SELECT 3
SET FILTER TO ORDERNR = ORDERS->ORDERNR    && Filtervoorwaarden instellen in subtabellen
SELECT 1
GO TOP
```

Hoofdstuk

20

Recordinformatie zoeken en samenvatten

U gebruikt zoekcommando's en -functies om bepaalde gegevens in een tabel te bekijken of te wijzigen. dBASE verschaft een aantal methoden waarmee u met of zonder index records zoekt. U kunt echter meestal sneller zoeken op basis van een index.

In dit hoofdstuk worden programmeertechnieken beschreven voor het zoeken naar gegevens. Zie Hoofdstuk 2 in het *Handboek* voor een beschrijving van de basisprincipes van het zoeken naar records en het vervangen van gegevens.

In dit hoofdstuk worden de volgende onderwerpen behandeld:

- LOCATE gebruiken
- Geïndexeerde records zoeken met FIND, SEEK en SEEK()
- Gegevens samenvatten en berekeningen uitvoeren
- Filters en weergaven gebruiken

Records zoeken

dBASE kent commando's en functies waarmee u geïndexeerd kunt zoeken en die waarden als resultaat geven. U kunt ook zonder index zoeken met het commando LOCATE of de functie LOOKUP(). Bij LOOKUP() hebt u alleen profijt van de uitgebreide functionaliteit als u geïndexeerd zoekt.

Opmerking Bij zoekacties wordt de verwerking van tekenreeksen waarin speciale tekens (tekens met accenten) voorkomen, onder meer beïnvloed door de huidige taalaansturing. Zie Appendix C voor meer informatie.

Zoeken met LOCATE

Met LOCATE kunt u een tabel recordsgewijs doorzoeken zonder indexen te gebruiken. Dit is een gemakkelijke, flexibele en efficiënte zoekmethode voor kleine tabellen.

In het volgende voorbeeld ziet u hoe LOCATE wordt gebruikt om het telefoonnummer van een klant in een tabel op te zoeken.

```
USE KLANTEN
LOCATE FOR NAAM = "Helder "      && Klant zoeken op basis van deel van de naamreeks
IF FOUND()
   DISPLAY telefoon              && Telefoonnummer weergeven in het uitvoerpaneel
                                 && van het Commandovenster
ENDIF
```

Zoeken met FIND en SEEK

Bij FIND en SEEK wordt een sleutelvolgorde gebruikt, zodat snel naar gegevens kan worden gezocht in en geïndexeerde tabel. SEEK heeft hierbij de voorkeur, vooral in eigen programma's. FIND biedt compatibiliteit met eerdere versies van dBASE.

FIND heeft de volgende kenmerken:

- De functie maakt gebruik van een teken-, datum- of numerieke index om het teken, de reeks of het getal te zoeken dat u hebt opgegeven.
- Een deelzoekreeks moet beginnen met het meest linkse teken.
- U kunt de gezochte tekenreeksen niet tussen scheidingstekens plaatsen.
- U kunt verkorte indexsleutels gebruiken als het indexlabel van het gegevenstype tekenreeks is en SET EXACT is ingesteld op OFF. Een deelreeks moet beginnen met het meest linkse teken.
- U kunt niet naar namen van geheugenvariabelen zoeken. Wel kunt u de macrofunctie & gebruiken.
- Er wordt rekening gehouden met hoofdletters en kleine letters. Als het indexlabel UPPER() bevat, kunt u niet zoeken naar reeksen met kleine letters. Gebruik functies voor reeksconversie, zoals UPPER() of LOWER(), om de hoofdletters en kleine letters in de zoekreeks overeen te laten komen met die in het indexlabel.

SEEK heeft de volgende eigenschappen:

- De functie maakt gebruik van indexen die zijn gebaseerd op uitdrukkingen waarin u één of meer velden of operatoren kunt opnemen die tekens, getallen of datums combineren.
- U kunt de gezochte tekensreeksen tussen scheidingstekens plaatsen.
- U kunt verkorte indexsleutels gebruiken als het indexlabel van het gegevenstype tekenreeks is en SET EXACT is ingesteld op OFF. Een deelreeks moet beginnen met het meest linkse teken.
- U kunt naar namen van geheugenvariabelen zoeken.
- Er wordt rekening gehouden met hoofdletters en kleine letters. Als het indexlabel UPPER() bevat, kunt u niet zoeken naar reeksen met kleine letters. Gebruik functies

voor reeksconversie, zoals UPPER() of LOWER(), om de hoofdletters en kleine letters in de zoekreeks overeen te laten komen met die in het indexlabel.

De functie SEEK() heeft dezelfde eigenschappen als het commando SEEK en resulteert daarnaast in True of False voor de zoekbewerking. U kunt daarom beter de functie SEEK() gebruiken dan FIND of SEEK.

Als geen index is geopend, werkt de functie LOOKUP() op dezelfde manier als LOCATE en wordt de tabel recordsgewijs doorzocht. U gebruikt LOOKUP() dan echter niet op een doeltreffende manier. Als u een geïndexeerd bestand gebruikt bij LOOKUP(), kunt u een uitdrukking zoeken in een veld en gegevens terughalen uit een ander veld.

Gebruik bijvoorbeeld SEEK en LOOKUP() in een geïndexeerde zoekactie waarbij u aan het veld AFNEMERNR gerelateerde informatie wilt zoeken.

```
USE AFNEMERS
INDEX ON AFNEMERNR TAG Afnemernr
SEEK 7845                                        && Bepaalde klant zoeken op basis van
                                                 && afnemernummer
Bedrag=LOOKUP(OPENBALANS,afnemernr,AFNEMERNR)    && Startbalans ophalen voor afnemernummer
? Bedrag                                      && Informatie over balans van afnemer teruggeven

Naam = LOOKUP(Bedrijf, afnemernr, AFNEMERNR)  && Bedrijfsnaam van afnemernummer zoeken
                                                 && Naam van overeenkomende afnemer teruggeven
                                              && Slaat afnemernummer op naar geheugenvariabele
```

Met het commando SEEK en de functie SEEK() kunt u ook zoeken op basis van indexsleutels. De functie SEEK() geeft een logische waarde terug voor het zoekresultaat.

```
CLOSE ALL
USE AFNEMERS
SET ORDER TO naam                 && Hoofdindex gebruiken
SEEK("Helder Water")              && Exacte zoekreeks opgeven
DISPLAY TELEFOON, AFNEMERNR       && In het bestand geselecteerde veldnamen
                                  && weergeven in resultaatvenster
```

De zoekreeks moet exaxt overeenkomen, inclusief hoofdletters en kleine letters. Anders resulteert SEEK() in False.

```
? SEEK("helder WAter")              && Geeft .F. terug
```

Wisselwerking SET NEAR en zoeken

Met de omgevingsinstelling SET NEAR wijzigt u het resultaat van de functies FOUND() en EOF() als bij het zoeken geen exacte overeenkomst wordt gevonden. In Tabel 20.1 ziet u een overzicht van de resultaatwaarden.

Tabel 20.1 SET NEAR en de resultaatwaarden als zoeken met zoekfuncties mislukt

	SET NEAR ON	SET NEAR OFF
Positie recordaanwijzer	Op record na dichtstbijzijnde overeenkomst	Aan einde van bestand
EOF()	False	True

Tabel 20.1 SET NEAR en de resultaatwaarden als zoeken met zoekfuncties mislukt (vervolg)

	SET NEAR ON	SET NEAR OFF
FOUND()	False	False
SEEK()	False	False

Het feit dat EOF() resulteert in True als SET NEAR is uitgeschakeld, kunt u vaak gebruiken om het zoekresultaat te bepalen. Er kunnen echter ook zoekbewerkingen voorkomen waarbij EOF() True teruggeeft terwijl SET NEAR wel is ingesteld op ON. Als het laatste record in het bestand bijvoorbeeld "B" is en u zoekt naar "C", wordt de recordaanwijzer op het record na de dichtstbijzijnde overeenkomst geplaatst, hetgeen overeenkomt met EOF().

Zoekresultaat bepalen

Als de zoekbewerking met SEEK is voltooid, gaat u aan de hand van de resultaatwaarden van de functies FOUND() en EOF() na of het zoeken is gelukt. U kunt ook alleen de functie SEEK() gebruiken om het zoekresultaat te bepalen.

De functie SEEK() kan hetzelfde resultaat opleveren als de combinatie van IF FOUND() en EOF(), omdat ook hiermee de recordaanwijzer wordt verplaatst en de logische waarde True of False wordt verkregen. U kunt het zoekresultaat het beste bepalen met de functie SEEK() omdat u met FOUND() aanvullende code moet invoeren om het logische zoekresultaat te testen. In het volgende voorbeeldprogramma wordt een bestaand record gezocht dat vervolgens wordt weergegeven. Als u SET NEAR ON gebruikt en de zoekbewerking mislukt, wordt de recordaanwijzer op het record geplaatst dat het meest voldoet aan de zoekreeks. Dit record wordt vervolgens weergegeven.

Het gebruik van de constructie IF en IF .NOT. levert de waarde op die ook met FOUND() zou zijn verkregen.

```
IF SEEK("W8662")
   DISPLAY                        && Overeenkomend record weergeven in het
                                  && resultaatpaneel van het commandovenster
ENDIF
SET NEAR ON
IF .NOT. SEEK("A8330")
   DISPLAY                        && Best overeenkomend record weergeven
                                  && in het resultaatpaneel van het commandovenster
ENDIF
```

Recordgegevens samenvatten

Als u berekeningen uitvoert, kunt u numerieke velden gebruiken of een uitdrukking (ook rekenveld genoemd) definiëren op basis van één of meer velden. De waarde van een rekenveld wordt niet opgeslagen en blijft slechts in het geheugen zolang de tabellen zijn geopend en er een berekening wordt uitgevoerd. U kunt deze velden alleen lezen en niet bewerken.

De tabel ARTIKEL.DBF bevat bijvoorbeeld onder andere de velden Artnr, Eenhprijs en Aantal bevat en de tabel OMZET.DBF de velden Artnr en Verknr bevat. Het volgende programmagedeelte toont hoe u rekenvelden maakt.

```
USE ARTIKEL ORDER Artnr
USE OMZET ORDER Verknr IN SELECT()
SET RELATION TO Artnr INTO OMZET
SET FIELDS TO OMZET->Verknr, Ext_Bedr=Aantal*Eenhprijs
```

Het rekenvak heet Ext_Bedr. Dit veld blijft bestaan zolang de tabellen zijn geopend en gerelateerd of totdat u SET FIELDS uitschakelt.

Rekenen met veldwaarden

Hoewel u meervoudige berekeningen ook kunt uitvoeren met afzonderlijke commando's zoals SUM of AVERAGE, kunt u hiervoor beter alleen het commando CALCULATE gebruiken, omdat u dan hetzelfde commando voor verschillende berekeningen kunt gebruiken. Met CALCULATE verkrijgt u betere prestaties omdat de records maar één keer worden verwerkt, ongeacht de hoeveelheid verschillende typen berekeningen die u opgeeft.

Bijvoorbeeld:

```
CALCULATE AVG(KOSTEN), STD(KOSTEN) TO mGem_kosten, mStd_kosten
```

In het volgende voorbeeld wordt het aantal orders per klant geteld. Allereerst wordt de tabel Orders geïndexeerd op basis van het veld Klantnr. Vervolgens kan de gebruiker een klantnummer invoeren voor de routine. SEEK wordt gebruikt om het klantnummer in het bestand te zoeken, waarna het aantal orders (records) van deze klant wordt geteld. Bovendien wordt berekend hoeveel deze klant gemiddeld heeft besteld.

```
mKlant = SPACE(6)                                   && Zes spaties om het klantnummer in
                                                    && te voeren
INDEX ON KLANTNR TAG Klantnr                        && Tabel indexeren
@ 10,10 SAY "Voer klantnummer in: " GET mKlant      && Melding weergeven en invoer
                                                    && accepteren
READ                                                && Ingevoerd klantnummer lezen
SET KEY TO mKlant
SEEK mKlant                                         && Klantnummer zoeken
CALCULATE CNT(), AVG(AANT_ART) TO mOrder_tel, mGem_aantal;
  WHILE mKlant = KLANTNR                            && Aantal orders berekenen alsmede
                                                    && de gemiddelde bestelling van deze
                                                    && klant
```

Tabel 20.2 toont een overzicht van het gebruik van CALCULATE om te rekenen met waarden uit de tabel

Tabel 20.2 Tabelgegevens samenvatten

Taak	Voorbeeld
Elementen tellen	CALCULATE CNT() TO mTellen of COUNT TO mtellen

Tabel 20.2 Tabelgegevens samenvatten (vervolg)

Taak	Voorbeeld
Waarden optellen	CALCULATE SUM(mkosten*maantal) TO mTotaal of SUM mkosten*maantal TO mtotaal
Gemiddelde van waarden	CALCULATE AVG(Kosten) TO mGem_Kosten of AVERAGE Kosten TO mGem_Kostent
Standaarddeviatie en variantie berekenen	CALCULATE STD(mKosten) TO mStd_Kosten en CALCULATE VAR(mKosten) TO mVar_Kosten
Minimumwaarde berekenen[1]	CALCULATE MIN(Kosten) TO mMin_Kosten of ? MIN(mDatum1,mDatum2)
Maximumwaarde berekenen[1]	CALCULATE MAX(Datum_trans) TO mMax_Datum of ? MAX(mAangen,mOntslag)

1. MIN() en MAX() resulteren bij CALCULATE in de kleinste of grootste waarde in het benoemde veld. Als u deze functies zelfstandig gebruikt, geven deze de kleinste of grootste waarde van twee waarden terug.

Records filteren in Query-ontwerp

U kunt Query-ontwerp gebruiken om interactief filtervoorwaarden te maken op basis van velden in een tabel. In Query-ontwerp wordt het commando SET FILTER gegenereerd om de records te selecteren die in de tabel worden verwerkt.

1 Open Query-ontwerp. Dubbelklik op het pictogram **Query's** of kies **Bestand | Nieuw | Query**.

2 Open het bestand dat u in de filtervoorwaarde wilt opnemen. U ziet het schema van het bestand in Query-ontwerp. (Als u gegevens uit meerdere tabellen wilt weergeven, kiest u **Tabel toevoegen** in het menu **Query** om aanvullende bestandsschema's weer te geven.

3 Geef in de ruimte onder de veldnaam in het schema de filtervoorwaarden op met behulp van relationele operatoren en uitdrukkingen . U kunt hiervoor ook het voorwaardenvak in Query-ontwerp gebruiken.

4 Selecteer **Query-resultaten** om het resultaat van de filter te bekijken. De gegevens in de tabel worden weergegeven in de opgegeven volgorde.

Aangezien in Query-ontwerp programmacode wordt gemaakt voor de query die u hebt ontworpen, kunt u deze code in andere programma's overnemen om hetzelfde resultaat te bereiken. In het volgende voorbeeld ziet u de code die door Query-ontwerp wordt gegenereerd voor een query op basis van één tabel:

```
CLOSE DATABASES
SET EXACT ON
SELECT 1
USE BEDRIJF.DBF
```

```
SET FILTER TO REGIO = 'NW'
GO TOP
```

Zie het *Handboek* en Help voor meer informatie over Query-ontwerp.

Velden selecteren

Gebruik SET FIELDS TO <*veldenlijst*> om alleen de velden weer te geven met de gegevens die u wilt bekijken of om velden uit andere werkgebieden toe te voegen. Gebruik SET FIELDS ON en SET FIELDS OFF om te schakelen tussen een eigen veldenlijst en de standaardveldenlijst van dBASE met alle velden. Als u een query ontwerpt met Query-ontwerp, wordt het commando SET FIELDS TO gegenereerd voor de velden die u in de query wilt opnemen.

Als u SET FIELDS TO gebruikt zonder opties, wordt de volledige veldenlijst weergegeven in plaats van de beperkte veldenlijst.

Hoofdstuk

21

Programmeren voor een netwerkomgeving

Als gegevens worden gedeeld, kunnen er conflicten ontstaan als meerdere gebruikers op hetzelfde moment dezelfde gegevens proberen bij te werken. dBASE beschikt over een ingebouwde beveiliging waarmee u dergelijke problemen kunt voorkomen. Als u toegang hebt tot bepaalde gegevens, kunt u records en tabellen automatisch of expliciet vergrendelen voordat u de gegevens bijwerkt.

In dit hoofdstuk worden de basismethoden beschreven die u kunt gebruiken als u gegevens weergeeft en bijwerkt in een netwerkomgeving. U kunt ook werken met transacties. Bij transacties worden pas wijzigingen in een tabel doorgevoerd als is vastgesteld dat de bijwerkingsprocedure succesvol is verlopen. Zie Hoofdstuk 22 voor meer informatie over transacties.

In dit hoofdstuk worden dBASE-bewerkingen beschreven die betrekking hebben op het gezamenlijk gebruik van gegevens in een netwerk De uitleg is echter ook van toepassing op het bewerken van gedeelde gegevens in verschillende *instanties* van dBASE of in verschillende programma's die in dezelfde instantie worden gestart.

De bewerkingen waarmee u in dBASE gegevens bijwerkt, gelden voor dBASE-, Paradox- en SQL-tabellen. De commando's en de functies die in dit hoofdstuk worden besproken, zijn eveneens op deze tabeltypen van toepassing, tenzij anders wordt vermeld. (Zie Hoofdstuk 23 voor meer informatie over het verschil in het gebruik van commando's en functies van dBASE als u Paradox- en SQL-tabellen inleest.)

Gegevens openen in een gezamenlijke omgeving

In dBASE voor Windows is sprake van een gezamenlijke omgeving in de volgende situaties:

- Gegevens zijn opgeslagen op een netwerkstation en worden gezamenlijk gebruikt.

- U hebt op uw computer meerdere instanties van dBASE gestart en gebruikt daarbij dezelfde gegevens in meerdere instanties.
- U hebt één instantie van dBASE gestart en opent dezelfde gegevens in afzonderlijke programma's binnen deze instantie.

U kunt op de volgende manieren op dezelfde computer gedeelde gegevens openen:

- Stel de omgevingsvariabele LOCALSHARE in het bestand IDAPI.CFG in op ON.
- Geef de DOS-opdracht SHARE op (niet vereist bij Windows voor Workgroups of als u in een netwerk werkt).
- Als u Paradox-tabellen wilt openen, stelt u eerst de directory van het netwerkbestand in met het configuratieprogramma IDAPI. (In dBASE voor Windows worden Paradox-vergrendelingen voor alleen-lezen niet ondersteund als Paradox-tabellen worden geopend.)

Als u een tabel opent met het commando USE, wordt de tabel automatisch geopend voor gedeeld gebruik. U wijzigt deze standaardinstelling door **Exclusief** in te schakelen in het menu **Kenmerken | Bureaublad**. Als u bij elk afzonderlijk bestand wilt bepalen of dit exclusief wordt geopend, gebruikt u de optie EXCLUSIVE bij het commando USE. Deze procedure wordt in de volgende sectie beschreven.

Ook de instelling **Sessies** bepaalt hoe tabelgegevens worden gedeeld. Een sessie in één instantie van dBASE voor Windows staat gelijk met één gebruikerssessie in een multi-user omgeving. Elke sessie beheert de eigen verzameling van 255 werkgebieden. De bewerking van een bestand in een sessie heeft geen invloed op andere sessies en u hebt vanuit andere sessies geen toegang tot een bestand dat exclusief is geopend.

De optie **Sessies** is standaard ingeschakeld. U wijzigt deze instelling in het menu **Kenmerken | Bureaublad** of met het commando CREATE SESSION in een programma.

Zie "Werken met sessies" verderop in dit hoofdstuk voor meer informatie over sessies.

Tabellen openen voor exclusief gebruik

Als u USE gebruikt met het sleutelwoord EXCLUSIVE, wordt een tabel exclusief geopend. Dit betekent dat alleen u toegang hebt tot de tabel, totdat u deze sluit met CLEAR ALL, CLOSE ALL, CLOSE TABLES, USE of QUIT.

```
USE NAMEN EXCLUSIVE          && Tabel openen voor exclusief gebruik
```

Als u een tabel probeert te openen die exclusief is geopend door een andere gebruiker of applicatie, of als u een bestand exclusief probeert te openen dat al exclusief wordt gebruikt door een andere gebruiker of applicatie, verschijnt de melding dat de tabel al in gebruik is.

U kunt een aantal dBASE-commando's alleen in tabellen gebruiken die exclusief zijn geopend. Het betreft de volgende commando's:

CONVERT

COPY INDEXES

DELETE TAG

INDEX ON...TAG

INSERT [BLANK]

MODIFY STRUCTURE

PACK

REINDEX

RESET

ZAP

Open een tabel exclusief als u bijvoorbeeld de tabelstructuur wilt wijzigen (met MODIFY STRUCTURE). Als u de tabel niet exclusief hebt geopend en gegevens uit die tabel worden weergegeven of bijgewerkt door andere gebruikers of applicaties, ontstaat er een conflict zodra u de tabelstructuur wijzigt of velden toevoegt of verwijdert.

Tabellen openen voor alleen lezen

Als u de optie NOUPDATE opgeeft bij USE, kunt u de tabel alleen lezen. Dit betekent dat het niet mogelijk is om records aan de tabel toe te voegen of records te wijzigen of te verwijderen.

```
USE OMZET NOUPDATE          && Tabel openen voor alleen lezen
```

Ook toegangsrechten die in een netwerk zijn ingesteld voor tabellen en databases hebben gevolgen voor de bewerkingen die u kunt uitvoeren.

Gegevens vergrendelen

Als een tabel door meerdere gebruikers of applicaties wordt bewerkt, kan er een conflict ontstaan als verschillende gebruikers of applicaties dezelfde gegevens proberen bij te werken. Met dBASE kunt u dergelijke conflicten voorkomen door slechts één gebruiker of applicatie tegelijk gegevens te laten bijwerken. Als een gebruiker de gegevens dan bijwerkt, wordt de tabel of het record in kwestie vergrendeld. Andere gebruikers of applicaties kunnen de gegevens wel weergeven, maar kunnen de tabel of het record pas bijwerken of vergrendelen als de eerste gebruiker klaar is.

U kunt bewerkingen in dBASE op twee manieren vergrendelen: *automatisch* en *expliciet*. Met automatische vergrendeling worden tabellen en records automatisch vergrendeld als deze worden bijgewerkt. Met expliciete vergrendeling kunt u een volledige tabel vergrendelen of alleen bepaalde records die u wilt bijwerken.

Automatische vergrendeling

Een record of een tabel wordt in dBASE automatisch vergrendeld als u BROWSE of EDIT gebruikt en daarbij met het toetsenbord of de muis elke willekeurige bewerking uitvoert waarmee records worden gewijzigd. Dit geldt niet voor de bewerkingen waarmee u door de tabel navigeert of een opdracht in een menu kiest. (U kunt een record ook vergrendelen met **Tabel | Record vergrendelen** of met *Ctrl-L* als de recordaanwijzer

zich op een record bevindt dat niet is vergrendeld.) Het record dat u wilt bijwerken, wordt dan vergrendeld samen met gerelateerde records in andere tabellen.

Paradox-tabellen worden in dBASE op dezelfde manier vergrendeld als dBASE-tabellen: het record wordt vergrendeld en bijgewerkt en vervolgens wordt de vergrendeling opgeheven. In SQL-tabellen worden de bewerkingen op een bepaald record eerst opgenomen, waarna wordt gecontroleerd of de bewerking op het record een conflict veroorzaakt met het record dat is opgeslagen in de database-server van waaruit de SQL-tabel is geopend. Als er geen conflict is, wordt het record in de database-server bijgewerkt. (Raadpleeg de documentatie bij Borland SQL Link voor uw database-server voor meer informatie over het gebruik van vergrendelingen in uw omgeving.)

Als een record of een tabel is vergrendeld, kunt u de gegevens bewerken. Anders verschijnt de melding dat het record of de tabel door een andere gebruiker wordt bewerkt. Als u de tabel met het commando CONVERT hebt geconverteerd (zie "Extra informatie over recordvergrendelingen opvragen"), wordt tevens de naam van de gebruiker weergegeven die de tabel of het record heeft vergrendeld. In beide gevallen gaat de vergrendelingspoging door totdat deze is gelukt of totdat u **Annuleren** kiest.

Als u een record hebt vergrendeld waarvan de gegevens zijn gewijzigd nadat u het voor het laatst hebt weergegeven, worden de gegevens bijgewerkt en verschijnt de melding **Record is door een ander gewijzigd**. Kies **OK** als u het bijgewerkte record wilt bewerken.

Als u een commando uitvoert waarmee de gehele tabel wordt gewijzigd, wordt de tabel automatisch vergrendeld. In Tabel 21.1 ziet u een overzicht van de commando's die een automatische vergrendeling veroorzaken (als er geen expliciete vergrendeling is). Hierbij wordt aangegeven of deze de volledige tabel vergrendelen of alleen bepaalde records. Als u bij DELETE, RECALL of REPLACE een recordnummer opgeeft met *<bereik>*, wordt niet de tabel maar het record automatisch vergrendeld..

Tabel 21.1 Commando's die een automatische vergrendeling veroorzaken

Commando	Type vergrendeling
APPEND	Record
APPEND FROM	Tabel
AVERAGE	Tabel
BLANK	Record
BLANK *<bereik>*	Tabel
BROWSE[1]	Record
CALCULATE	Tabel
CHANGE[1]	Record
COPY	Tabel
COPY STRUCTURE	Tabel
COUNT	Tabel
DELETE	Record
DELETE *<bereik>*	Tabel
EDIT[1]	Record
INDEX	Tabel
JOIN	Tabel

Tabel 21.1 Commando's die een automatische vergrendeling veroorzaken (vervolg)

Commando	Type vergrendeling
LABEL FORM	Tabel
RECALL	Record
RECALL *<bereik>*	Tabel
REPLACE	Record
REPLACE *<bereik>*	Tabel
REPORT FORM	Tabel
SORT	Tabel
SUM	Tabel
TOTAL	Tabel
UPDATE	Tabel

1. dBASE vergrendelt het record als u op een toets drukt waarmee de gegevens zouden worden gewijzigd.

Als u een record hebt bijgewerkt of een ander record actief maakt, wordt de tabel of het record automatisch vrijgegeven zodat het door andere gebruikers of applicaties kan worden bewerkt. U kunt de vergrendeling ook handmatig opheffen door **Tabel | Vergrendeling opheffen** te kiezen of op *Ctrl-L* te drukken als de recordaanwijzer zich op een vergrendeld record bevindt.

U kunt ook het commando UNLOCK gebruiken om de vergrendeling van records of tabellen in bepaalde of in alle werkgebieden op te heffen. (Zie "Tabellen en records expliciet vergrendelen" verderop in dit hoofdstuk.)

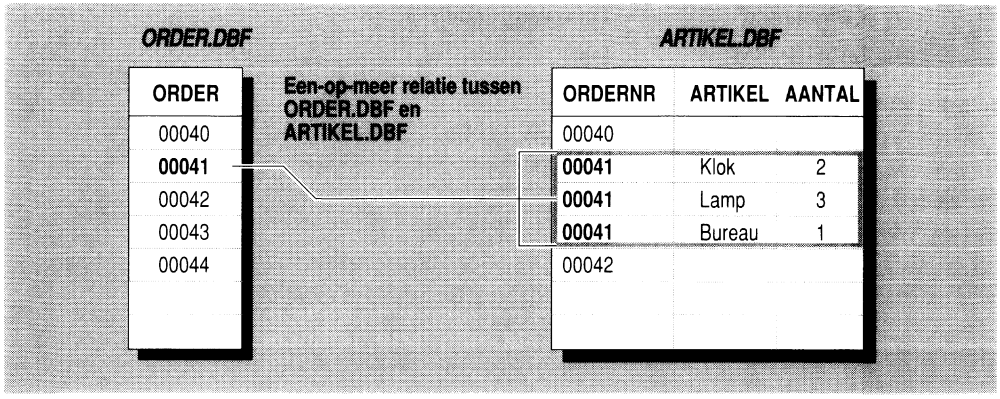
Gerelateerde tabellen automatisch vergrendelen

Als u gegevens bewerkt waarbij velden uit meerdere tabellen zijn betrokken (met SET RELATION of SET VIEW), worden automatisch de bijbehorende gerelateerde records in elke tabel vergrendeld. U kunt bijvoorbeeld de volgende relatie instellen tussen de tabellen Orders en Artikelen:

```
USE ORDERS IN SELECT() ALIAS Orders                          && Hoofdtabel openen
USE ARTIKELEN IN SELECT() ORDER Factnr ALIAS Artikelen       && Subtabel openen
SELECT Orders
SET RELATION TO Ordernr INTO Artikelen                       && Orders van elke klant
                                                             && koppelen
SET SKIP TO Artikelen                                        && Door orders van elke klant
                                                             && navigeren
SET FIELDS TO ORDER->ORDERNR, ARTIKELEN->ARTK_BSCHR, ARTIKEL->AANTAL
BROWSE
```

U hebt nu een één-op-meer-relatie gedefinieerd tussen de orders en de artikelen in elke order. Als u door de orders navigeert en vervolgens de afzonderlijke artikelen in een bepaalde order bewerkt, wordt het huidige record in de subtabel vergrendeld. Als u de recordaanwijzer naar de volgende order verplaatst, wordt de vergrendeling opgeheven. In Afbeelding 21.1 ziet u hoe deze bewerking wordt uitgevoerd:

Afbeelding 21.1 Gerelateerde records vergrendelen



U bewerkt bijvoorbeeld de velden van het record met ordernummer 00041. Het desbetreffende record in de tabel Order en het huidige record in de tabel Artikel worden dan vergrendeld. De vergrendeling van beide tabellen wordt opgeheven zodra u de recordaanwijzer naar een ander record verplaatst.

Automatische vergrendeling uitschakelen en inschakelen

U kunt besluiten de automatische vergrendeling in of uit te schakelen, afhankelijk van de hoeveelheid gebruikers of sessies die een bestand in gebruik hebben of van de duur of de aard van de bewerking. Een vergrendeling voorkomt dat er conflicten in de gegevensbewerking ontstaan, maar heeft wel tot gevolg dat deze vertraagd wordt uitgevoerd. Als u een tabel bijwerkt die slechts zelden wordt gebruikt, kunt u de vergrendeling uitschakelen, zodat u de bewerking sneller kunt uitvoeren.

Gebruik SET LOCK om de automatische vergrendeling in of uit te schakelen. In Tabel 21.2 ziet u een overzicht van de commando's waarvoor u de automatische vergrendeling kunt uitschakelen.

Tabel 21.2 Commando's waarmee u de automatische vergrendeling uitschakelt

Commando's	
AVERAGE	JOIN
CALCULATE	LABEL FORM
COPY	REPORT FORM
COPY STRUCTURE	SORT
COUNT	SUM
INDEX...TO <*.NDX-bestand*>	TOTAL

De vergrendeling is standaard ingeschakeld. Als u de vergrendeling voor deze commando's wilt uitschakelen, gebruikt u SET LOCK OFF voordat u een commnado opgeeft.

Let op Als de automatische vergrendeling is uitgeschakeld, is gegevensintegriteit niet gegarandeerd.

Bij sommige van deze commando's heeft SET LOCK OFF alleen effect als u gegevens uit een tabel leest , niet als u naar de tabel schrijft. Als u naar een tabel schrijft, wordt bij COPY, COPY STRUCTURE, INDEX, JOIN, SORT en TOTAL de doeltabel automatisch geopend voor exclusief gebruik.

Extra informatie over recordvergrendeling opvragen

Als u het commando CONVERT opgeeft, wordt een veld _dbaselock toegevoegd aan een bestaande tabel. In dit veld wordt statusinformatie opgeslagen over de vergrendeling en de bijwerking van een record. De beschikbare informatie is afhankelijk van de veldlengte die u opgeeft bij CONVERT. Bijvoorbeeld:

```
USE KLANTEN                    && Tabel KLANTEN.DBF openen
CONVERT TO 24                  && Veld maken met een lengte van 24 tekens
```

Als het veld _dbaselock is gemaakt, wordt voor elk record de volgende informatie opgeslagen:

- De datum en het tijdstip waarop het record voor het laatst is vergrendeld.
- Of het record is gewijzigd nadat u dit voor het laatst hebt weergegeven.
- De volledige of gedeeltelijke naam (afhankelijk van de lengte van het veld CONVERT) van de gebruiker die het record het laatst heeft vergrendeld of gewijzigd.

Opmerking Voordat het nieuwe veld _dbaselock wordt toegevoegd, maakt het commando CONVERT eerst een reservekopie met de extensie .CVT van de originele tabel. De tijd en de ruimte die nodig is om de geconverteerde tabel te maken, is afhankelijk van de omvang van de tabel.

Met LKSYS() kunt u de datum en het tijdstip bepalen waarop het record voor het laatst is vergrendeld. Als u hierbij een veldlengte opgeeft van meer dan acht tekens, wordt ook de naam van de gebruiker weergegeven die het record voor het laatst heeft vergrendeld. CHANGE() resulteert in .T. als het record in de tabel is bijgewerkt nadat u de gegegevens voor het laatst hebt weergegeven en in .F. als de gegevens niet zijn gewijzigd.

Als een record is gewijzigd, gebruikt u REFRESH om de gegevens in de tabel bij te werken. U kunt ook met SET REFRESH een interval opgeven voor de tijdstippen waarop het scherm automatisch wordt bijgewerkt met de gegevens die op dat moment in de tabel zijn opgeslagen. Bijvoorbeeld:

```
USE KLANTEN                    && Tabel KLANTEN.DBF openen
SET REFRESH TO 1000            && Interval instellen voor automatisch bijwerken van scherm
IF CHANGE()                    && Controleren of het huidige record is bijgewerkt
   REFRESH                     && Nieuwe gegevens weergeven indien bijgewerkt
ELSE
   REPLACE...                  && Recordwaarden bijwerken indien gegevens niet
                               && zijn gewijzigd
ENDIF
```

Tabellen en records expliciet vergrendelen

Dankzij automatische vergrendeling kunt u met een minimale hoeveelheid programmeerwerk een single-user-applicatie overbrengen naar een multi-user- of gezamenlijke omgeving. U kunt echter ook automatische en expliciete vergrendeling van tabellen en records combineren in uw applicaties. De kans op een *botsing* van gegevens, die zich voordoet als twee of meer gebruikers tegelijkertijd dezelfde gegevens proberen te wijzigen, wordt verder verkleind als u tabellen en records expliciet vergrendelt. Bovendien wordt een expliciete vergrendeling meestal sneller opgeheven, zodat meer gebruikers toegang kunnen hebben tot de tabelgegevens.

U kunt definiëren hoe de wisselwerking tussen uw applicatie en de gebruikers verloopt en bepalen welke mogelijkheden twee of meer gebruikers hebben met tabellen of records die zijn vergrendeld. Er kan zich bijvoorbeeld een *impasse* voordoen als twee gebruikers geen toegang kunnen krijgen tot tabellen of records die beiden nodig hebben om een transactie uit te voeren. Er onstaat dan een oneindige lus, waarin beide gebruikers wachten totdat de ander de vergrendeling opheft. U doorbreekt deze impasse van herhaalde pogingen door uw applicatie te wijzigen zodat de ene gebruiker eerst toegang krijgt tot alle tabellen en records die nodig zijn om een transactie uit te voeren, en daarna de andere gebruiker. (Zie Hoofdstuk 22 voor meer informatie over transacties.)

FLOCK() en RLOCK()

U gebruikt de functies FLOCK() en RLOCK() om te bepalen of een tabel of een record is vergrendeld door een andere gebruiker of applicatie. Als dit niet het geval is, wordt de tabel of het record vergrendeld. U gebruikt FLOCK() om de volledige tabel te vergrendelen en RLOCK() of LOCK() om het huidige record te vergrendelen samen met alle records (in gerelateerde tabellen) die afhankelijk zijn van het huidige record.

Als de tabel die u wilt vergrendelen nog niet is vergrendeld door een andere gebruiker, wordt de tabel of het record zowel met FLOCK() als met RLOCK() vergrendeld, waarna .T als resultaat wordt gegeven. Als de tabel wél is vergrendeld of als de vergrendelingsbewerking om een bepaalde reden mislukt, resulteren beide functies in .F.

Als u SET REPROCESS gebruikt in combinatie met deze twee functies, kunt u het aantal vergrendelingspogingen opgeven dat wordt gedaan als de vergrendeling niet onmiddellijk kan worden ingesteld. U kunt met SET REPROCESS drie verschillende typen waarden instellen:

- Als u de waarde instelt op -1, wordt de poging om een tabel te vergrendelen of openen (met het commando USE), herhaald totdat een andere gebruiker die de tabel of het record heeft vergrendeld, de vergrendeling opheft. Er kan dan een impasse ontstaan omdat er geen melding verschijnt en andere gebruikers de herhalingsprocedure niet kunnen beëindigen.
- Als u de waarde instelt op 0, worden de vergrendelingspogingen onbeperkt herhaald. Er verschijnt echter een dialoogvenster, zodat de gebruiker op *Esc* kan drukken om de vergrendelingspoging te beëindigen.

- Als u een positieve waarde opgeeft, wordt de vergrendelingspoging gestaakt na het opgegeven aantal pogingen.

Het volgende voorbeeld toont hoe u in een applicatie expliciete vergrendeling gebruikt in combinatie met SET REPROCESS:

```
SET REPROCESS TO 100          && Aantal pogingen instellen
IF RLOCK()
   <bijwerken>
    :
ENDIF
```

In dit voorbeeld worden er honderd vergrendelingspogingen gedaan voordat wordt overgegaan naar IF...ENDIF. U kunt ook een programa maken waarin u SET REPROCESS gebruikt met de routine ON ERROR. De routine ON ERROR die u definieert, wordt pas uitgevoerd als de lus van SET REPROCESS is beëindigd. De routine ON ERROR wordt niet uitgevoerd als u bij SET REPROCESS de waarde -1 opgeeft.

Expliciete vergrendeling van tabel of record opheffen

U kunt de vergrendeling van een tabel of een record op de volgende twee manieren opheffen:

- Met UNLOCK heft u de vergrendeling van tabellen of records op in alle of in bepaalde werkgebieden.
- Met CLOSE ALL, CLOSE TABLES, USE, CLEAR ALL of QUIT sluit u een tabel.
- Zie *Commando's en functies* voor meer informatie over het gebruik van deze commando's.

Programmeren voor netwerkomgevingen

Gebruik NETWORK() in uw applicaties om de omgeving te testen en te bepalen of de applicatie op een netwerk wordt uitgevoerd. Op die manier kunt u het gedrag van het progamma aanpassen aan de vraag of het gegevens opent in een netwerk of wordt gebruikt in een exclusieve single-user-omgeving.

```
IF NETWORK()                  && Controleren of programma op een netwerk wordt uitgevoerd
   Lname = ID()               && Aanmeldingsnaam van huidige netwerkgebruiker teruggeven
ENDIF
```

Als de gegevens bijvoorbeeld vanuit een netwerkomgeving worden geopend, kunt u bijvoorbeeld naast optimale tabel- en recordvergrendeling, verschillende typen foutverwerkingsroutines definiëren (met het commando ON NETERROR). Als u SQL-tabellen opent, kunt u de applicatie aanpassen door foutverwerkingsroutines te definiëren voor de database-server waarmee u werkt.

Werken met sessies

Met sessies kunt u verschillende omgevingsinstellingen en resources verbinden met dBASE-objecten, zoals formulieren, programma's, BROWSE- en EDIT-vensters, die worden geopend in de Windows-omgeving. Elke sessie verschaft een eigen verzameling werkgebieden en omgevingsvariabelen voor SET. U kunt in elke sessie instellingen voor opties wijzigen, transacties definiëren, tabellen en indexen openen en door records navigeren, zonder dat u hiermee bewerkingen beïnvloedt in andere sessies die op hetzelfde moment actief zijn.

Opmerking Een geheugenvariabele houdt in elke sessie dezelfde waarde.

Een sessie wordt echter wel beïnvloed door bijwerkingen en tabel- en record-vergrendelingen die in andere geopende sessies worden uitgevoerd. Sessies hebben een zelfde soort gedrag als gebruikers die in een netwerkomgeving gegevens openen. Als een tabel of een record in de ene sessie is vergrendeld, kan de tabel of het record niet worden vergrendeld in de andere sessie. Als een tabel in de ene sessie wordt bijgewerkt, worden in andere sessies de gegevens in dezelfde tabel bijgewerkt zodra de sessie weer actief wordt.

Sessies maken

De instellingen en bewerkingen voor een sessie worden gebaseerd op de keuzes die u maakt op het moment dat u de sessie maakt of gebruikt. Met de optie **Sessies** in **Kenmerken | Bureaublad** bepaalt u of sessies worden gebruikt als u een tabel of een query opent met Navigator of met **Bestand | Openen**. Rapporten en labels worden altijd in een eigen sessie bewerkt.

U kunt in een programma het commando CREATE SESSION opnemen (meestal aan het begin van het programma) om een nieuwe sessie te starten. (Zie *Commando's en functies* voor meer informatie over CREATE SESSION en voor een overzicht van de omgevingsinstellingen, de commando's en de functies die door een sessie worden bestuurd.) Als u een nieuwe sessie maakt, worden alle waarden bij SET ingesteld op de oorspronkelijke opstartwaarden (uit DBASEWIN.INI gelezen). De tabellen in de andere actieve sessies blijven geopend en in de nieuwe sessie zijn geen tabellen geopend en wordt het huidige werkgebied ingesteld op 1.

```
CREATE SESSION                  && Nieuwe sessie starten
USE KLANTEN                     && Tabel openen
BROWSE                          && BROWSE-venster openen
CREATE SESSION                  && Tweede sessie maken
USE KLANTEN                     && KLANTEN.DBF opnieuw openen
SKIP 1                          && Navigatie in tweede sessie heeft geen invloed op
                                && recordaanwijzer in eerste sessie
```

In Formulierontwerp voegt u het commando CREATE SESSION toe aan de gegenereerde code voor een formulier als u dit in een eigen sessie wilt uitvoeren.

Opmerking Het commandovenster wordt altijd in een eigen sessie uitgevoerd, zodat u elke bewerking in het commandovenster onafhankelijk kunt uitvoeren van de overige sessies waarin u bijvoorbeeld met Navigator een object hebt geopend, zoals een query of een tabel.

Sessies selecteren

Er is niet een bepaald commando waarmee u een sessie selecteert. U selecteert een sessie als u deze maakt of als u een venster van de sessie actief maakt. Een sessie die is verbonden met een formulier wordt actief als u een methode of een actieverwerking voor het formulier start. Als u een andere geopende sessie actief maakt, worden alle omgevingsinstellingen hersteld die u hebt gedefinieerd toen u deze sessie maakte.

Sessies sluiten

Een sessie blijft geopend zolang een programma, een formulier, BROWSE- of EDIT-venster waarin naar de sessie wordt verwezen, actief is. Als er geen enkele entiteit meer actief is waarin naar de sessie wordt verwezen, wordt de sessie met de daarin geopende tabellen automatisch gesloten. Als er een transactie wordt uitgevoerd, verschijnt er een dialoogvenster waarin u de transactie kunt doorvoeren (commit), terugdraaien (rollback) of annuleren.

Hoofdstuk

22

Werken met transacties

De applicaties die u uitvoert in een netwerkomgeving, zijn meestal gecompliceerder en krijgen met meer gevarieerde situaties te maken dan de applicaties die u in een single-user omgeving uitvoert. De bewerkingen kunnen bijvoorbeeld trager worden uitgevoerd of worden onderbroken als verschillende gebruikers dezelfde tabel gezamenlijk gebruiken. De tabellen die u inleest met een applicatie op een netwerk zijn bovendien meestal groter en bevatten vaak gegevens die algemeen worden gebruikt.

In dit hoofdstuk wordt beschreven hoe u transacties gebruikt om de integriteit van uw gegevens te handhaven. Als u transacties gebruikt, kunt u eerst controleren of de records succesvol zijn bijgewerkt voordat u de wijzigingen doorvoert in de tabel. U voorkomt dan dat de integriteit en de betrouwbaarheid van de gegevens verloren gaat door gedeeltelijk gelukte of mislukte wijzigingen van de gegevens.

De bewerkingen die in dit hoofdstuk worden beschreven, gelden zowel voor dBASE-, Paradox- als SQL-tabellen, tenzij anders wordt vermeld. Zie Hoofdstuk 23 voor meer informatie over de specifieke verschillen in het gebruik van de commando's en de functies van dBASE waarmee u Paradox- en SQL-tabellen inleest.

Transactieverwerking

Als u gegevens bijwerkt, is het vaak belangrijk te weten of u de bewerking kunt voltooien en de wijzigingen daadwerkelijk kunt aanbrengen. Dit is bijvoorbeeld van belang als u meerdere records of tabellen tegelijk bewerkt. In een transactie wordt een verzameling bewerkingen als een eenheid gegroepeerd, zodat alle wijzigingen in één keer worden doorgevoerd, of geannuleerd als de transactie om een bepaalde reden niet kan worden voltooid.

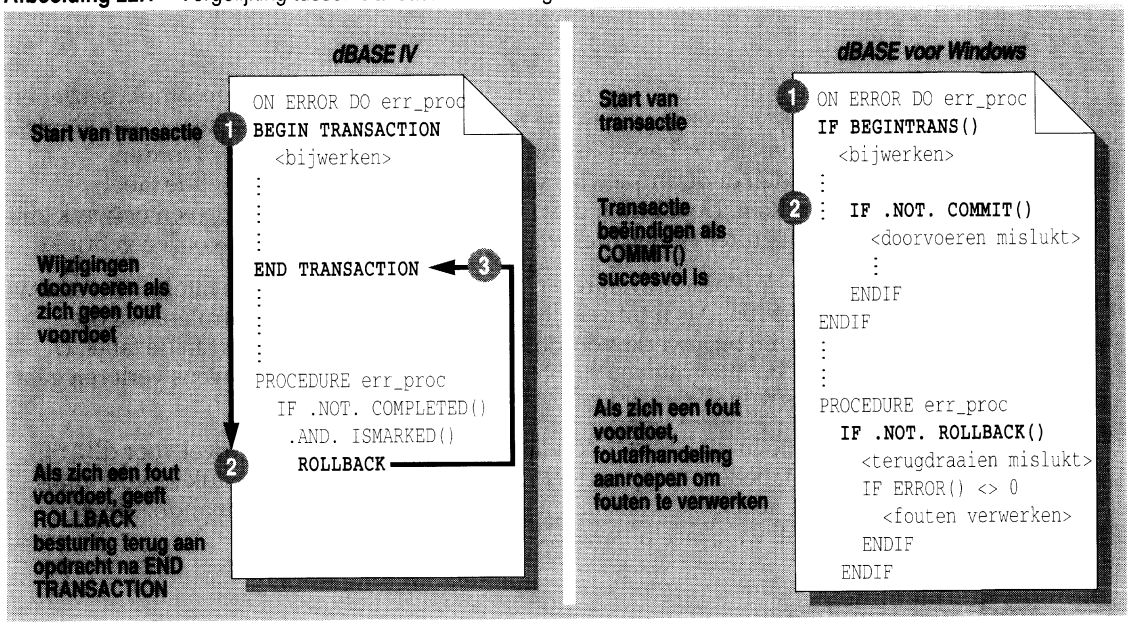
Stel, dat u records in een invoerformulier voor meerdere tabellen bijwerkt of records in batch bijwerkt met het commando REPLACE. U definieert dan een transactie waarin alle bijwerkingen worden opgenomen die het gevolg zijn van uw bewerking. Als het dan niet lukt om met de transactie alle records bij te werken (als u bijvoorbeeld vergrendelde records of tabellen wilt bijwerken), kunt u de gegevens weer in de

originele staat weergeven. U kunt dan later opnieuw proberen om de transactie te voltooien.

Transacties in dBASE IV en dBASE voor Windows

U kunt transacties verwerken in zowel dBASE IV als dBASE voor Windows. Er worden hierbij wel verschillende methoden gebruikt. De transactieverwerking in dBASE IV is *blok-* of *batch-georiënteerd*, terwijl de transacties in dBASE voor Windows *actiegestuurd* zijn. Met deze methode kunt u transacties *doorvoeren* of *terugdraaien*, onafhankelijk van de code of de dBASE-opdrachten waarmee u de transactie hebt gestart. Hierdoor kunt u de transactieverwerking gemakkelijker gebruiken in actiegestuurde applicaties, die u bijvoorbeeld in dBASE ontwikkelt en uitvoert onder Windows. Afbeelding 22.1 toont deze bewerking aan de hand van een eenvoudig voorbeeld.

Afbeelding 22.1 Vergelijking tussen transactieverwerking in dBASE IV en dBASE voor Windows



Bij de definitie van een transactie in dBASE IV worden alle bewerkingen die nodig zijn om gegevens bij te werken, geplaatst in een blok tussen de opdrachten BEGIN TRANSACTION en END TRANSACTION. De transactie wordt doorgevoerd als de opdracht END TRANSACTION wordt uitgevoerd. Als zich een fout voordoet tijdens de transactie en de wijzigingen worden teruggedraaid met een foutafhandelingsroutine, gaat de uitvoering van het programma verder met de opdracht direct na END TRANSACTION.

In dBASE worden alle bewerkingen na BEGINTRANS() opgenomen in de transactie, ongeacht de wijzigingen of de vertakkingen die zich in het programmaverloop voordoen. De transactie wordt beëindigd zodra ROLLBACK() of COMMIT() wordt uitgevoerd. De uitvoering van het pogramma gaat dan verder vanaf dat punt. Dit is met name handig als u een programma maakt waarin formulieren worden gebruikt. U zorgt

dan dat de wijzigingen pas worden doorgevoerd als de gebruiker de recordaanwijzer naar een ander veld of het volgende record verplaatst of het formulier sluit. (Zie "Transacties gebruiken in formulieren", verderop in dit hoofdstuk.)

Transacties maken

dBASE biedt u een verzameling functies, BEGINTRANS(), COMMIT() en ROLLBACK(), waarmee u dBASE-, Paradox- en SQL-tabellen kunt bijwerken met transactieverwerkingen. U kunt de wijzigingen doorvoeren of terugdraaien en de tabel weer in de originele staat weergeven, afhankelijk van het resultaat van de transactie.

BEGINTRANS() start een transactie. De functie geeft .T. terug als de transactie succesvol is gestart. De records en de tabellen blijven vergrendeld totdat de transactie wordt beëindigd.

COMMIT() beëindigt een transactie. Hiermee wordt elke wijziging die u in tabellen of indexen hebt aangebracht, definitief ingevoerd. Deze functie heft tevens alle record- en tabelvergrendelingen op en geeft .T. terug als de transactie succesvol is doorgevoerd.

ROLLBACK() beëindigt een transactie eveneens. Hiermee wordt elke wijziging in geopende tabellen en indexen ongedaan gemaakt, waarna alle record- en tabelvergrendelingen worden opgeheven. De functie geeft .T. terug als de transactie succesvol is teruggedraaid.

```
ON ERROR ROLLBACK()                          && Wijzigingen ongedaan maken als zich
                                             && een fout voordoet
USE RECTELS
IF BEGINTRANS()                              && Transactie proberen te starten
   REPLACE ALL Verkprijs WITH Verkpijs * 1.1 && Wijziging die ROLLBACK()kan
                                             && terugdraaien
   IF .NOT. COMMIT()                         && Wijzigingen proberen door te voeren
      *Doorvoeren mislukt; opties voor gebruikers maken
      *voor nieuwe poging of bijwerken annuleren
      :
   ENDIF
ENDIF
```

Transacties verwerken voor dBASE- en Paradox-tabellen

Als u een Paradox- of dBASE-tabel inleest, wordt de procedure waarmee u wijzigingen doorvoert of terugdraait, uitgevoerd door dBASE. Als de transactie geldt voor een tabel die is opgeslagen op een databaseserver, worden de transactiecommando's naar de server gestuurd en wordt de transactieprocedure uitgevoerd door de server. Er kunnen dus andere regels gelden voor lokale transacties of voor transacties op andere servers. De transacties waarmee u Paradox- of dBASE-tabellen bijwerkt, worden uitgevoerd door dBASE.

Opmerking U kunt geen transactie definiëren waarmee u zowel een dBASE- of Paradox-tabel als een SQL-tabel bijwerkt. U kunt transacties evenmin nesten. Als u SQL-gegevens inleest, kunt u wél afzonderlijke transacties voor elke geopende database definiëren, omdat elke

databaseserver de transacties zelf uitvoert. Zie Hoofdstuk 23 voor meer informatie over transacties definiëren om SQL-tabellen bij te werken.

Commando's en functies van dBASE gebruiken in transacties

U kunt alle commando's en functies waarmee u tabellen bijwerkt, opnemen in een dBASE-transactie. De volgende tabel toont een overzicht van de commando's en de functies van dBASE die u in een transactie kunt opnemen.

Tabel 22.1 Commando's en functies die zijn toegestaan in dBASE-transacties

Commando's en functies	
APPEND [BLANK]	BLANK
APPEND MEMO	INSERT
REPLACE	BROWSE
REPLACE MEMO	EDIT
REPLACE BINARY	@ GET...READ
REPLACE OLE	RLOCK()
DELETE	FLOCK()
RECALL	

De bewerkingen die u uitvoert met het commando APPEND FROM worden niet opgenomen in een dBASE-transactie.

In de volgende tabel ziet u de commando's die niet zijn toegestaan in een dBASE-transactie.

Tabel 22.2 Commando's en functies die niet zijn toegestaan in dBASE-transacties

Commando's en functies	
BEGINTRANS()	CREATE FROM
CLEAR ALL	DELETE TAG
CLOSE ALL	INSERT
CLOSE DATABASE	MODIFY STRUCTURE
CLOSE INDEX	PACK
CLOSE TABLES	ZAP
CONVERT	

Evenals in een transactie van dBASE IV, heeft het commando UNLOCK pas effect als een transactie in dBASE voor Windows is beëindigd. Bovendien zijn geneste transacties noch in dBASE IV noch in dBASE voor Windows toegestaan.

Opmerking U kunt PACK gebruiken om records uit een dBASE-tabel te verwijderen. Met DELETE kunt u alleen aangeven welke records u wilt verwijderen zonder deze daadwerkelijk uit de tabel te verwijderen.

Transacties gebruiken in formulieren

In eerdere versies van dBASE gebruikte u commando's als @...SAY...GET..., APPEND, BROWSE en REPLACE om gegevens in een tabel bij te werken. In Formulierontwerp kunt u echter de kenmerken DataSource en DataLink gebruiken om formulieren te maken waarvan u de elementen, zoals invoervakken, direct koppelt aan de tabelgegevens. Als u het formulier vervolgens opent, worden de gegevens automatisch weergegeven in de invoervakken. Als u de gegevens van het formulier bewerkt, wordt het bijbehorende record in de onderliggende tabel vergrendeld en bijgewerkt overeenkomstig de ingevoerde wijzigingen.

Transactiepunten definiëren

U kunt transacties, evenals andere dBASE-commando's waarmee u gegevens bijwerkt, ook als onderdeel van een formulier of menu gebruiken. Een voorbeeld hiervan zijn de procedures WeergaveBewerk en ControleerCommit in het formulier KLANTEN.WFM, een voorbeeldbestand dat bij de installatie van dBASE is inbegrepen. In dit formulier wordt een transactie gestart als de gebruiker de opdracht Bewerken kiest in het menu KLANTEN.MNU en klantgegevens wijzigt.

```
procedure WeergaveBewerk

local inBewerkModus, bewerkMenu, stuurelement
   bewerkMenu = form.basis.klanten.WeergaveBewerk
   *** Transactie sluiten bij verlaten bewerkmodus, anders openen
   if form.inBewerkModus
      form.ControleerGewijzigd(.f.)
      bewerkMenu.text = "Be&werken"
      bewerkMenu.shortcut = "Ctrl-W"
      bewerkMenu.statusMessage = "Gegevens bewerken."
      form.basis.klanten.verwijderen.enabled = .f.
                                              && toegestaan in weergavemodus
      form.ControleerCommit(.f.)              && Transactie controleren en
      form.text = "Klanten -- weergavemodus"  && naar weergavemodus
      form.notitiesEditor.colorNormal = "N/W"
      form.childBlader.modify = .f.
      form.notitiesEditor.modify = .f.
      form.statusmessage = "Weergavemodus. " + ;
                           "Kies Klanten | Bewerken (Ctrl-W) om " + ;
                           "gegevens te bewerken/verwijderen."
      form.volgendeKlantKnop.SetFocus()
   else                                       && Naar bewerkmodus
      bewerkMenu.text = "&Weergave"
      bewerkMenu.shortcut = "Ctrl-W"
      bewerkMenu.statusMessage = "Gegevens weergeven."
      form.basis.klanten.verwijderen.enabled = .t.
                                              && toegestaan in bewerkmodus
      form.ControleerCommit(.t.)              && Transactie controleren en
      form.text = "Klanten -- bewerkmodus"    && naar bewerkmodus
      form.childBlader.modify = .t.
      form.notitiesEditor.modify = .t.
```

```
      form.statusmessage = "Bewerkmodus. " + ;
                           "Kies Klanten | Weergave (Ctrl-W) om " + ;
                           "gegevens te bekijken."
      form.naamInvoer.SetFocus()       && Naar Naam-invoervak
   endif
   inBewerkModus = form.inBewerkModus  && Het is sneller om in een lus niet
                                       && steeds een formuliervariabele
   stuurelement = form.first           && uit te hoeven lezen
   do
      if .not. stuurelement.className $;
"BROWSE,EDITOR,VORIGEKNOP,VOLGENDEKNOP,PUSHBUTTON,IMAGE,TEXT"
         stuurelement.enabled = inBewerkModus
      endif
      stuurelement = stuurelement.before
   until stuurelement.name = form.first.name
   form.klantnrInvoer.enabled = .f.      && Sleutelveld is altijd geblokkeerd
   form.naamInvoer.SetFocus()            && Naar Naam-invoerveld
```

WeergaveBewerk roept eerst ControleerGewijzigd aan, waarin wordt gecontroleerd of de gebruiker de gegevens heeft bewerkt. Vervolgens wordt ControleerCommit aangeroepen, waarin een transactie wordt gestart met de functie BEGINTRANS().

```
procedure ControleerCommit (newInBewerkModus)

private orderVeld, wijzGemaakt
   wijzGemaakt = form.wijzGemaakt
   if form.wijzGemaakt .and. form.inBewerkModus
      orderVeld = field(1)                  && Veld Klantnummer
      if ConfirmationMessage("Wijzigingen doorvoeren?",;
                             ReeksOpmaak("Klant %1",&orderVeld)) = YES
         commit()
      else
         rollback()
         if .not. empty(form.vorigeRecord)
            go form.vorigeRecord
            form.vorigeRecord = .f.
         endif
      endif
      if form.inBewerkModus .and. newInBewerkModus
         begintrans()
      endif
      form.wijzGemaakt = .f.
   endif
   if form.inBewerkModus <> newInBewerkModus
      if newInBewerkModus                   && Naar BewerkModus
         begintrans()
      else                                  && Naar WeergaveModus
         if .not. wijzGemaakt
            rollback()
         endif
      endif
      form.inBewerkModus = newInBewerkModus
   endif
```

Als de gebruiker het formulier heeft bewerkt, verschijnt een dialoogvenster waarin de gebruiker opgeeft of de wijzigingen moeten worden doorgevoerd, teruggedraaid of geannuleerd. Dit dialoogvenster verschijnt ook als u dBASE afsluit terwijl een transactie actief is.

Er zijn nog meer manieren om in dBASE wijzigingen te evalueren die u uitvoert met een transactie. U kunt bijvoorbeeld het kenmerk OnChange gebruiken met gegevenselementen, zoals invoervakken, keuzelijsten met invoervakken, en vervolgens controleren of een veld is gewijzigd.

```
DEFINE ENTRYFIELD KLANTNRINVOER OF THIS;
  PROPERTY;
  :
  DataLink "Klantnr", ;
  OnChange CLASS::WIJZGEMAAKT, ;
  :
```

U kunt de methode OnNavigate definiëren om te controleren of een gebruiker de recordaanwijzer heeft verplaatst en welke wijzigingen deze in het record heeft aangebracht. Als het formulier wordt gesloten, kunt u ook controleren welke wijzigingen in de tabel zijn aangebracht.

```
CLASS KLANTEN OF FORM;
  this.OnOpen=CLASS::FORM_OPEN
  this.OnClose=CLASS::FORM_ONCLOSE
  this.OnNavigate CLASS::WIJZGEMAAKT
  :
```

U kunt de transactieverwerking ook toepassen op formulieren waarvoor u de commando's APPEND of REPLACE gebruikt in plaats van objecten (zoals invoervakken) of methoden (zoals OnChange of OnNavigate) om een record toe te voegen of bij te werken of batch-bewerkingen uit te voeren waarmee u meerdere records in een tabel bijwerkt.

Hoofdstuk

23

Werken met niet-dBASE tabellen

Dankzij Borland Database Engine heeft u met dBASE automatisch toegang tot dBASE- en Paradox-tabellen. Als u Borland SQL Link hebt geïnstalleerd, kunt u bovendien gegevens uit SQL-tabellen inlezen.

U kunt in Paradox- en SQL-tabellen gebruik maken van commando's en functies van dBASE. Met de functie SQLEXEC() kunt u bovendien rechtstreeks SQL-opdrachten uitvoeren om gegevens uit dBASE-, Paradox- of SQL-tabellen in te lezen.

In dit hoofdstuk worden de verschillen uitgelegd tussen het inlezen van dBASE-, Paradox-, en SQL-tabellen in dBASE. Verder wordt beschreven hoe u programmeert om de verschillende tabeltypen aan te passen.

Zie het *Handboek* als u wilt weten hoe Borland Database Engine door dBASE wordt gebruikt om verschillende tabeltypen in te lezen. Raadpleeg de documentatie bij Borland SQL Link voor meer informatie over SQL-tabellen gebruiken in dBASE.

U kunt andere gegevensbestanden inlezen door deze in een dBASE-tabel te importeren. Zie de commando's IMPORT en APPEND FROM in *Commando's en functies*.

Verschil tussen Paradox- en SQL-tabellen

Als u bewerkingen uitvoert in een niet-dBASE tabel, worden de regels gehanteerd die gelden voor het desbetreffende tabeltype. Dit geldt bijvoorbeeld voor de benoeming en de samenstelling van indexsleutels. Een indexsleutel van dBASE kan bijvoorbeeld bestaan uit een samengestelde uitdrukking, waarin velden, functies en operatoren worden gecombineerd, zoals in UPPER (Afdeling+Titel). Een indexsleutel in een Paradox- of SQL-tabel kan echter alleen uit een veld of een veldenlijst bestaan. Het is daarom van belang dat u weet welke regels gelden voor het tabeltype dat u gebruikt.

Paradox- en SQL-tabellen kunnen veldtypen bevatten die niet door dBASE-tabellen worden ondersteund. U kunt lees- en schrijfbewerkingen uitvoeren op deze velden als u Navigator gebruikt of dBASE-commando's en -functies uitvoert om gegevens in te lezen. Zie de documentatie van Paradox voor meer informatie over de gegevenstypen

van Paradox. Zie de documentatie van Borland SQL Link voor de databaseserver waarmee u werkt, voor meer informatie over de gegevenstypen van SQL-tabellen.

Als u gegevens van een niet-dBASE veldtype opslaat in een geheugenvariabele, worden de gegevens geconverteerd naar een compatibel gegevenstype. Er wordt bijvoorbeeld een tekenvariabele gemaakt als u een alfanumeriek veld uit Paradox toewijst aan een geheugenvariabele.

Omgekeerd worden de gegevens eveneens geconverteerd naar het niet-dBASE type als u een geheugenvariabele plaatst in een veld van een niet-dBASE tabel. U kunt bijvoorbeeld een alfanumeriek veld uit Paradox vervangen door de inhoud van een tekenvariabele.

Database openen

Een *database* is een logische groepering van tabellen, opgeslagen in een bepaalde directory op een bepaalde databaseserver waar u toegang tot hebt met Borland SQL Link. U kunt een database maken of definiëren met het configuratieprogramma IDAPI. Zie *Aan de slag* voor meer informatie.

Als u een database hebt gedefinieerd, gebruikt u eerst het commando OPEN DATABASE om een verbinding te maken met de database of de bijbehorende databaseserver, voordat u de tabellen inleest. U gebruikt SET DATABASE om de huidige database in te stellen als er meerdere databases zijn geopend. U kunt met dBASE eveneens de lokatie van de tabel in een database opgeven door, voorafgaand aan de naam van de tabel, de naam van de database te typen tussen dubbele punten. Bijvoorbeeld:

```
USE :BoekhDB:NWORDERS          && Tabel Nworders openen in database BoekhDB
```

Als u een database opgeeft die niet is geopend, verschijnt er een dialoogvenster waarin u informatie, zoals aanmeldingsnaam en wachtwoord, invoert om een verbinding te maken met de database.

Tabeltype opgeven

Als u een tabel opent met USE, plaatst u de extensie .DBF of .DB achter de naam van de tabel om respectievelijk een dBASE- of een Paradox-tabel te openen. Als u geen extensie opgeeft, wordt het tabeltype geopend dat bij SET DBTYPE is ingesteld (de standaardinstelling is dBASE). Het gebruik van de optie TYPE bij het commando USE gaat voor de instelling bij SET DBTYPE.

In een database wordt het type geopend dat hierin is opgeslagen. Dit kan een SQL-, Paradox- of dBASE-tabel zijn.

```
USE KLANTEN                    && dBASE-tabel KLANTEN.DBF openen
USE ORDERS.DB                  && Paradox-tabel ORDERS.DB openen
USE ARTIKEL TYPE PARADOX       && Paradox-tabel ARTIKEL.DB openen
SET DBTYPE TO PARADOX          && Tabeltype standaard instellen op Paradox
USE ORDERS                     && Paradox-tabel ORDERS.DB openen
USE KLANTEN                    && Poging om KLANTEN.DBF te openen mislukt
```

```
USE KLANTEN TYPE DBASE          && dBASE-tabel KLANTEN.DBF openen
COPY TO NIEUW                   && Naar Paradox-tabel NIEUW.DB kopiëren
USE :BoekhDB:NWKLANT            && Tabel NWKLANT openen in database BoekhDB
```

SET DBTYPE heeft ook invloed op het commando DIR.

```
DIR                             && Overzicht van .DBF-bestanden weergeven
SET DBTYPE TO PARADOX
DIR                             && Overzicht van .DB-bestanden weergeven
```

Veldnamen tussen scheidingstekens plaatsen

Veldnamen in Paradox-tabellen en andere niet-dBASE tabellen kunnen spaties en andere tekens, zoals rechte haakjes ([ ]), een hekje (#) of haakjes, bevatten. Dit heeft invloed op elke uitdrukking waarin een veldnaam wordt gebruikt. In de code plaatst u veldnamen die niet worden ondersteund door dBASE-tabellen, tussen dubbele punten. Dit geldt niet voor de veldnamen die voldoen aan de regels van dBASE. U kunt een dubbele punt als scheidingsteken gebruiken voor commando's of functies waarvoor u een veld of een veldenlijst kunt opgeven.

```
USE FACTUREN.DB TYPE PARADOX                        && Paradox-tabel openen
DISPLAY :Naam klant:, Telefoon, :(Bank)reknr:       && Ongeldige veldnamen tussen
                                                    && scheidingstekens plaatsen
BROWSE FIELDS :Naam klant:, Telefoon, :(Bank)reknr: && Ongeldige veldnamen tussen
                                                    && scheidingstekens plaatsen
```

Records identificeren met bladwijzers

Paradox- en SQL-tabellen hebben in tegenstelling tot dBASE-tabellen geen vaste recordnummers. Een record krijgt een andere positie als de volgorde van de records in de tabel wordt gewijzigd. Als u bijvoorbeeld een tabel met een klantenbestand in Paradox opent, is Aagje Alberts uit Zwijndrecht waarschijnlijk het eerste record als Namen de hoofdindex is, en het laatste record als Plaats de hoofdindex is.

U gebruikt *bladwijzers* om naar records in Paradox- en SQL-tabellen te verwijzen. Aangezien u de afzonderlijke records niet kunt identificeren op basis van de positie in de tabel, kunt u de records in de tabel aanduiden met een bladwijzer. U kunt met BOOKMARK() of RECNO() een bladwijzerwaarde opslaan in een geheugenvariabele.

BOOKMARK() geeft altijd een bladwijzer terug. RECNO() geeft een recordnummer terug voor dBASE-tabellen en een bladwijzer voor Paradox- en SQL-tabellen. U gebruikt bladwijzers op dezelfde manier als recordnummers.

```
USE FACTUREN.DB ORDER Factnr TYPE PARADOX   && Paradox-tabel openen
SEEK "1001"                                 && Record zoeken
bkRecord = BOOKMARK()                       && Bladwijzer opslaan
GO TOP                                      && Recordaanwijzer verplaatsen
                                            && naar eerste record
GO bkRecord                                 && Terugkeren naar factuurnummer 1001
```

Een bladwijzer is een speciaal gegevenstype dat u niet kunt afdrukken. De waarde ervan is te vergelijken met de referentievariabele van een object. U kunt relationele operatoren gebruiken om bladwijzers met elkaar te vergelijken. U kunt echter geen query maken op basis van de waarde van een bladwijzer of deze waarde wijzigen.

```
bkRecord1 = BOOKMARK()                  && Bladwijzer opslaan
SKIP 10                                 && Recordaanwijzer op nieuw record
bkRecord2 = BOOKMARK()                  && Tweede bladwijzer opslaan
? bkRecord2 > bkRecord1                 && Geeft .T. terug
? bkRecord1                             && Geeft "BladWijzer" weer
```

Als u bewerkingen uitvoert waarmee de volgorde van records wordt gewijzigd, zoals gegevens in een sleutelveld wijzigen of het commando SET ORDER geven, worden de huidige bladwijzerwaarden ongeldig.

Het ontbreken van vaste recordnummers heeft gevolgen voor een aantal commando's en functies die u voor Paradox- en SQL-tabellen gebruikt:

- BROWSE, LIST en DISPLAY tonen geen kolom voor het recordnummer.
- GO en GOTO accepteren bladwijzerwaarden. GO en GOTO geven een foutmelding terug als u een numerieke waarde opgeeft.
- RECNO() geeft een bladwijzer terug.
- Het sleutelwoord RECORD accepteert een bladwijzerwaarde bij elk commando waarvoor u *<bereik>* kunt opgeven.
- LOCK() en RLOCK() accepteren een bladwijzerslijst. U kunt echter geen bestands- of recordvergrendeling toepassen in SQL-tabellen. (U gebruikt het commando ON ERROR en de functies ERROR(), DBERROR() en SQLERROR() om te controleren of de tabel succesvol is bijgewerkt.)
- REPLACE vereist een unieke index bij databaseservers die geen unieke recordadressen verschaffen.

Records invoegen

Als u records toevoegt aan een dBASE-tabel, kunt u deze aan het einde toevoegen (met APPEND) of invoegen op de huidige positie van de recordaanwijzer (met INSERT). U kunt alleen records toevoegen aan Paradox- en SQL-tabellen. In Paradox-tabellen heeft INSERT *met* een index dezelfde werking als APPEND in een dBASE-tabel. Het record wordt dan "ingevoegd" in de index en toegevoegd aan het einde van de tabel.

In SQL-tabellen heeft een index geen gevolgen voor de wijze waarop u records aan een tabel toevoegt. Als u records invoegt in een SQL-tabel, wordt de plaats en de manier waarop nieuwe records worden opgeslagen, bepaald door de databaseserver waarmee u werkt.

Records verwijderen

U voert twee stappen uit om een record uit een dBASE-tabel te verwijderen. Allereerst markeert u het record dat u wilt verwijderen met DELETE. Vervolgens verwijdert u het record uit de tabel met PACK. (U kunt het commando PACK alleen gebruiken als de tabel exclusief is geopend.) U verwijdert de markering met het commando RECALL, zodat het effect van DELETE ongedaan wordt gemaakt.

In Paradox- en SQL-tabellen kunt u de te verwijderen records niet markeren, zodat u in deze tabellen maar één stap uitvoert om records te verwijderen. Als u een record verwijdert met DELETE, wordt dit definitief uit de tabel verwijderd. U kunt deze bewerking niet ongedaan maken.

Voor Paradox- en SQL-tabellen geldt het volgende:

- DELETE verwijdert een record definitief uit de tabel, waarna de recordaanwijzer naar het volgende record wordt verplaatst. (In SQL-tabellen wordt de recordaanwijzer verplaatst naar het volgende record dat u hebt teruggehaald uit de databaseserver.)
- DELETED() geeft altijd .F. terug.
- RECALL en PACK geven foutmeldingen terug.
- SET DELETED heeft geen effect.

U gebruikt het commando ZAP om alle records uit dBASE-, Paradox- en SQL-tabellen te verwijderen.

Gegevens controleren

U kunt voor elk veld in Paradox- en SQL-tabellen een validiteitscontrole opgeven. Validiteitscontroles zijn regels voor de waarden die een gebruiker in een veld kan invoeren. U kunt dan bijvoorbeeld een minimum-, maximum- of standaardwaarde voor een veld opgeven.

De validiteitscontroles van Paradox en SQL zijn te vergelijken met de validiteitsclausules die u met het kenmerk **Geldig** opgeeft voor invoerobjecten voor gebruikers (zoals invoervakken). Het verschil is dat deze direct zijn gekoppeld aan de velden in de tabel.

In dBASE worden de bestaande validiteitscontroles voor Paradox- en SQL-tabellen gehandhaafd. Als u gegevens bewerkt in een veld met een validiteitscontrole van een Paradox-of SQL-tabel, wordt de validiteitsvoorwaarde geëvalueerd en wordt een foutmelding teruggegeven als niet aan de voorwaarde wordt voldaan. U kunt de validiteitscontrole echter niet maken of wijzigen. Hiervoor opent u Paradox of gebruikt u de daartoe bestemde hulpmiddelen van de databaseserver waar de tabel is opgeslagen. Zie de documentatie van Paradox of de databaseserver voor informatie over de beschikbare typen validiteitscontrole en hoe u deze maakt.

De validiteitscontroles van Paradox worden opgeslagen in een .VAL-bestand met dezelfde naam als de tabel. Als u een Paradox-tabel in dBASE opent, wordt het .VAL-bestand, indien dit bestaat, automatisch geopend.

Transacties gebruiken

dBASE beschikt over een verzameling functies waarmee u transacties kunt uitvoeren om dBASE-, Paradox- en SQL-tabellen bij te werken.

- BEGINTRANS() start een transactie.
- COMMIT() voert de wijzigingen van de huidige transactie door.
- ROLLBACK() draait wijzigingen terug en geeft de tabel in de originele staat weer.

Zie Hoofdstuk 21 voor meer informatie over transacties definiëren in dBASE-programma's.

Als u een Paradox- of dBASE-tabel inleest, wordt de procedure waarmee u wijzigingen doorvoert of terugdraait, uitgevoerd door dBASE. Als de transactie geldt voor een tabel die is opgeslagen op een databaseserver, worden de transactiecommando's naar de server gestuurd en wordt de transactieprocedure uitgevoerd door de server.

Opmerking U kunt geen transactie definiëren waarmee u zowel een dBASE- of Paradox-tabel als een SQL-tabel bijwerkt. U kunt transacties evenmin nesten. Als u SQL-gegevens inleest, kunt u wél afzonderlijke transacties voor elke geopende database definiëren, omdat elke databaseserver de transacties zelf uitvoert. U geeft bij BEGINTRANS(), COMMIT() of ROLLBACK de naam van de database op samen met de transactiefunctie om te bepalen welke databaseserver de transactie uitvoert.

Het volgende voorbeeld toont hoe u een transactie kunt definiëren voor gegevens die zijn opgeslagen op een databaseserver:

```
ON ERROR DO errorProc                          && Foutafhandelingsroutine opgeven
OPEN DATABASE Boekh                            && Database openen
USE :Boekh:MIJNTABEL                           && Tabel openen in de database
IF BEGINTRANS("Boekh")                         && Servertransactie starten
   REPLACE ALL SALARIS WITH SALARIS * Inflatie && Alle records proberen bij te werken
   IF .NOT. COMMIT("Boekh")                    && Wijzigingen doorvoeren als geen
                                               && fout heeft plaatsgevonden
      *Doorvoeren mislukt
      :                                        && Opties voor gebruiker maken als
   ENDIF                                       && transactie mislukt
ENDIF
   :
PROCEDURE errorProc                            && Procedure definiëren voor
                                               && foutafhandeling
   IF .NOT. ROLLBACK("Boekh")                  && Poging tot terugdraaien bij fout
      * Terugdraaien mislukt
      IF DBERROR() <> 0                        && Controleren op fout vanaf de server
         * IDAPI-fout verwerken
         :
         IF SQLERROR() <> 0                 && Controleren op fout vanaf databaseserver
```

```
            * SQL-fout verwerken
          ENDIF
        ENDIF
      ENDIF
    RETURN
```

Werken met indexen

De indexen van Paradox-tabellen worden onderhouden door dBASE in de indeling die geldt voor dit tabeltype. (De indexen van SQL-tabellen worden onderhouden door de databaseserver waar de geopende tabel is opgeslagen.) Als u een Paradox-tabel bewerkt met indexcommando's en -functies, worden de indexbestanden van Paradox gebruikt.

Paradox-tabellen gebruiken de volgende indexbestanden:

- Primaire indexen (.PX). Deze index bevat de *primaire sleutel.* Een primaire sleutel is een uniek identificatiesymbool voor een record, dat te vergelijken is met het recordnummer van dBASE.

  U kunt een primaire index alleen maken op basis van de eerste velden in een tabelstructuur. Wellicht moet u de positie van velden wijzigen om de primaire index te maken. U gebruikt het sleutelwoord PRIMARY als u een primaire index maakt.

- Secundaire enkelvoudige indexen (.Xnn en .Ynn). Deze index is gebaseerd op één enkel veld. Als u een secundaire index maakt van een veld, moeten de labelnaam en de veldnaam exact overeenkomen (voor indexen met onderscheid tussen hoofd- en kleine letters). U kunt daarom de labelnaam weglaten als u de index maakt.

- Secundaire samengestelde indexen (.XGn en .YGn). Deze index is gebaseerd op twee of meer velden.

Zie de documentatie van Paradox voor meer informatie over Paradox-indexen.

Zie Hoofdstuk 2 in het *Handboek* of de definities in Help als u niet bekend bent met de indexterminologie (zoals *produktie-index, indexlabel, hoofdindex*).

De indexen van Paradox worden in dBASE op dezelfde manier behandeld als de indexlabels in een produktie-indexbestand van dBASE. Alle indexen van Paradox worden bijvoorbeeld automatisch geopend zodra u de tabel opent. Deze blijven geopend totdat u de tabel sluit. De primaire index van een tabel wordt de hoofdindex. Als er geen primaire index is, worden de records in de natuurlijke volgorde weergegeven.

U verwijst naar de secundaire indexen van Paradox door deze als labelnamen te gebruiken. U gebruikt bijvoorbeeld SET ORDER TO (in plaats van SET INDEX TO) om de hoofdindex op te geven. Als u de primaire index wilt selecteren als hoofdindex, gebruikt u SET ORDER TO zonder indexnaam.

In het volgende voorbeeld heeft de tabel Facturen een primaire index (Factnr) en een secundaire index (Code_Art):

```
USE FACTUREN TYPE PARADOX      && Volgorde tabel volgens FACTNR
SET ORDER TO Code_Art          && Volgorde tabel volgens CODE_ART
SET ORDER TO                   && Volgorde tabel volgens FACTNR
```

Als u SQL-tabellen inleest, geeft u de naam van een index op (met SET ORDER of USE...ORDER TAG) om de volgorde te besturen waarop de records uit een tabel worden teruggehaald. U kunt een Paradox- of SQL-index echter niet sluiten.

Verschillen tussen indexsleutels

Een Paradox-index kan zijn gebaseerd op één veld of een veldenlijst. Een sleutel die bestaat uit een veldenlijst, wordt een *samengestelde sleutel* genoemd. Als u een samengestelde sleutel wilt opgeven, plaatst u een komma achter elke veldnaam. U kunt geen uitdrukkingen gebruiken in de sleutels van Paradox.

```
USE KLANTEN.DB
INDEX ON PLAATS PRIMARY                    && Primaire sleutel maken
INDEX ON REGIO, LAND TAG Regland           && Secundaire samengestelde sleutel maken
```

Als u in een Paradox-tabel wilt zoeken op basis van een index met een samengestelde sleutel, kunt u alleen SEEK en SET KEY TO gebruiken. Andere zoekcommando's en -functies, zoals FIND, SEEK(), LOOKUP() en KEYMATCH(), accepteren geen argumenten met samengestelde sleutels.

```
USE KLANTEN.DB ORDER Regland               && Openen met samengestelde index
SEEK 'Corfu', 'Griekenland'                && Zoeken met sleutel op basis van twee velden
```

dBASE- indexen wijken verder opde volgende punten af van de indexen uit Paradox- en SQL-tabellen:

- In een Paradox-index kunt u noch voorwaardelijke sleutels noch aflopende sleutels gebruiken. De opties FOR en DESCENDING van INDEX, en de functies FOR() en DESCENDING() geven een foutmelding terug voor Paradox-indexen.
- Het indextype dat u kunt maken voor SQL-tabellen (met het commando dBASE INDEX), is afhankelijk van de mogelijkheden van de onderliggende databaseserver. U kunt met elke databaseserver, net als in Paradox, een index maken op basis van een veld of een veldenlijst. Bovendien kunt u met de meeste databaseservers oplopende en aflopende indexen maken. U kunt echter geen voorwaardelijke indexen maken.
- Met de primaire indexen van Paradox worden automatisch unieke sleutelwaarden toegewezen. (Een unieke index op een databaseserver heeft hetzelfde effect.) Stel dat u een index maakt op basis van Regio en dit veld bevat een record met de regio "Corfu". U kunt dan in het veld Regio geen record meer toevoegen met de naam "Corfu". In een samengestelde primaire sleutel (op basis van meerdere velden) kunt u criteria definiëren voor de toewijzing van unieke waarden, zodat u bijvoorbeeld zowel op Regio als op Land indexeert.

  De toewijzing van unieke sleutelwaarden geldt ook voor lege records. Een geïndexeerde tabel kan niet meer dan één leeg record bevatten omdat er anders een dubbele lege sleutelwaarde ontstaat.
- De primaire index van Paradox en de unieke index van SQL verschillen van de dBASE-index die u maakt met de optie UNIQUE. De optie UNIQUE zorgt dat dubbele sleutelwaarden niet worden opgenomen in een dBASE-index, maar voorkomt niet dat u dubbele sleutelwaarden maakt als u een identiek record toevoegt. U kunt UNIQUE niet gebruiken in Paradox-tabellen. Met de optie

INDEX...UNIQUE maakt u in SQL-tabellen een unieke index waarmee u **wél** voorkomt dat er dubbele sleutelwaarden worden ingevoerd. UNIQUE() geeft altijd .T. voor de primaire index van een Paradox-tabel en .F. voor een secundaire index. SET UNIQUE heeft geen effect op Paradox-indexen.

In dBASE-programma's wordt vaak APPEND BLANK en REPLACE gebruikt om een leeg record toe te voegen aan een tabel en gegevens in een veld in te voeren. In Paradox- en SQL-tabellen gebruikt u in dat geval APPEND AUTOMEM om te voorkomen dat u dubbele lege records maakt. Met APPEND AUTOMEM voegt u een nieuw record toe en plaatst u tegelijkertijd de waarden uit eerder gemaakte automem-variabelen in het veld. Zie *Commando's en functies* voor meer informatie over het gebruik van deze commando's.

SQL-opdrachten uitvoeren

U leest een SQL-tabel in met dezelfde commando's en functies van dBASE als een dBASE- of Paradox-tabel. U kiest eerst een database en opent vervolgens de tabel met het commando USE. Daarna kunt u de gegevens weergeven en bewerken.

Als u een SQL-tabel opent met USE, wordt vanaf de server een SQL-opdracht uitgevoerd om de records terug te halen. De teruggehaalde gegevens worden opgeslagen in een lokale buffer die wordt onderhouden door dBASE. U kunt de gegevens dan weergeven en bewerken. Aangezien dit extra lees- en schrijfbewerkingen vereist, kan de bewerking van grote SQL-tabellen veel tijd in beslag nemen.

Als u de gegevens niet op de server hoeft te bewerken, wordt de uitvoering versneld als u zelf SQL-opdrachten schrijft. U opent een tabel dan niet met USE, maar u gebruikt de dBASE-functie SQLEXEC() om SQL-opdrachten rechtstreeks naar de databaseserver te sturen.

Opmerking U kunt SQLEXEC() ook gebruiken om gegevens van dBASE- en Paradox-tabellen in te lezen. U kunt de SQL-opdrachten dan gebruiken met de SQL-commando's van de databaseserver Borland InterBase, die voldoet aan de ANSI-norm.

Als u een SQL-opdracht rechtstreeks naar de server stuurt, kunt u elke bewerking uitvoeren die wordt ondersteund door de databaseserver waar de tabel is opgeslagen. Bovendien kunt u dan een query maken van meerdere tabellen of voorwaarden opgeven om de hoeveelheid teruggehaalde gegevens te beperken. U kunt bijvoorbeeld de volgende opdrachten uitvoeren:

```
SET DBTYPE TO DBASE
SET DATABASE TO Boekh
ErrorCode = SQLEXEC(' "SELECT Bedrijf, Plaats FROM Bedrijf + ;
   WHERE Regio = "NW" ', "LokaleCo")
IF ErrorCode = 0
   SET DBTYPE TO DBASE
   USE LOKALECO
   LIST
ENDIF
```

In dit voorbeeld gebruikt u de opdracht SELECT om gegevens uit een SQL-tabel terug te halen en op te slaan in de dBASE-tabel "LokaleCo". U kunt deze "opname" van gegevens dan weergeven of gebruiken om bijvoorbeeld een dBASE-rapport te maken.

Overzicht van de ondersteuning van Paradox- en SQL-tabellen

Tabel 23.1 toont een overzicht van de wijze waarop dBASE Paradox- en SQL-tabellen ondersteunt

Tabel 23.1 Overzicht van de wijze waarop dBASE omgaat met Paradox- en SQL-tabellen

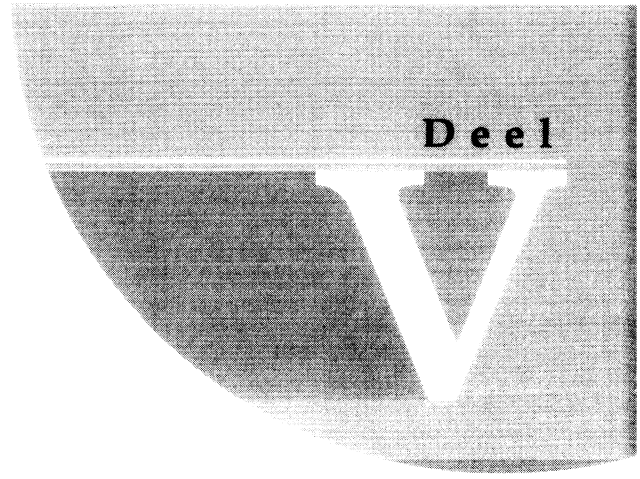
	Paradox- en SQL-tabellen	Gevolgen voor	dBASE voor Windows
Veldnamen	Veldnamen kunnen spaties en andere tekens bevatten die niet zijn toegestaan in veldnamen van dBASE.	Elke opdracht die een veldnaam gebruikt in een uitdrukking	Geeft een dubbele punt (:) als scheidingsteken voor veldnamen die niet voldoen aan de regels voor namen die in dBASE gelden.
Record-nummers	Geen vaste recordnummers.	LIST / DISPLAY	Toont geen kolom voor recordnummers bij Paradox- en SQL-tabellen.
		RECNO()	Geeft een recordnummer terug voor dBASE-tabellen en een bladwijzer voor Paradox- en SQL-tabellen.
		BOOKMARK()	Geeft een bladwijzer terug voor alle tabeltypen.
		GO / GOTO, LOCK(), RLOCK() Sleutelwoord RECORD voor commando's met <bereik>	Accepteert bladwijzers voor Paradox- en SQL-tabellen. Accepteert recordnummers of bladwijzers voor dBASE-tabellen.
		INSERT *zonder* primaire index	Bij SQL-tabellen is de werking afhankelijk van de server.
		INSERT *met* primaire index	Dezelfde werking als APPEND. Het record wordt ingevoegd in de index en toegevoegd aan het einde van de tabel.
Records verwijderen	Records worden direct uit de tabel verwijderd. U kunt dit niet meer ongedaan maken.	DELETE	Verwijdert records definitief uit de tabel.
		DELETED()	Geeft altijd .F. terug
		RECALL, PACK	Geeft fout terug.
		SET DELETED	Heeft geen effect.

Tabel 23.1 Overzicht van de wijze waarop dBASE omgaat met Paradox- en SQL-tabellen (vervolg)

	Paradox- en SQL-tabellen	Gevolgen voor	dBASE voor Windows
Index-bestanden	Alle geassocieerde indexen zijn geopend als de tabel is geopend en blijven geopend totdat de tabel wordt gesloten.	USE	Opent automatisch alle geassocieerde indexen (Welke vlag wordt "onderhouden" bij Paradox-tabellen, is niet van belang). De indexen blijven geopend totdat de tabel wordt gesloten.
	Verwijzen naar indexnamen en niet naar indexbestanden.	SET ORDER TO USE...ORDER TAG <labelnaam>	De hoofdindex wordt net als in een dBASE-tabel opgegeven met SET INDEX TO. U gebruikt de indexnaam als het label. Dit betekent dat u SET ORDER TO <indexnummer> niet kunt gebruiken.
		Optie PRIMARY van INDEX	U gebruikt de optie PRIMARY om te verwijzen naar primaire indexen in Paradox-tabellen. U gebruikt de indexnaam om naar secundaire indexen te verwijzen.
		SET INDEX TO Opties INDEX en OF van USE SET ORDER TO <indexnummer> COPY TAG COPY INDEXES Optie OF van DELETE TAG	Geeft altijd fout terug. U kunt alleen met labels verwijzen naar de indexen van Paradox- en SQL-tabellen.

Tabel 23.1 Overzicht van de wijze waarop dBASE omgaat met Paradox- en SQL-tabellen (vervolg)

	Paradox- en SQL-tabellen	Gevolgen voor	dBASE voor Windows
Index-sleutels	Een sleutel kan een enkel veld of een veldenlijst (een *samengestelde sleutel*) zijn. Uitdrukkingen zijn niet toegestaan.	INDEX Label maken met CREATE MODIFY STRUCTURE	Geeft fout terug als de sleutel een uitdrukking is. In samengestelde indexen gebruikt u een komma om de veldnamen van elkaar te scheiden.
		SEEK, SET KEY TO	Accepteert lijsten met samengestelde sleutels.
		FIND, SEEK(), LOOKUP(), KEYMATCH()	Gebruik van lijsten met samengestelde sleutels is niet toegestaan. U gebruikt SEEK en SET KEY TO om te zoeken op basis van indexen met samengestelde sleutels.
	In Paradox-tabellen zijn voorwaardelijke en aflopende sleutels niet toegestaan. (Aflopende indexen worden wel ondersteund door SQL-tabellen.)	Opties FOR en DESCENDING van INDEX	Geeft fout terug voor tabellen die geen voorwaardelijke en aflopende sleutels ondersteunen.
		FOR(), DESCENDING()	Geeft een lege reeks terug voor FOR(). DESCENDING() geeft .F. terug voor tabellen die geen aflopende index ondersteunen.
	Een primaire index wijst automatisch een unieke sleutelwaarde toe. Dit kan niet met een secundaire index. (In SQL-tabellen worden unieke indexwaarden toegewezen met een unieke index.)	Optie UNIQUE van INDEX	Geeft fout terug voor Paradox-tabellen. Voor SQL-tabellen wordt een unieke index gemaakt. Unieke sleutelwaarden worden automatisch toegewezen.
		UNIQUE()	Geeft altijd .T. terug voor primaire indexen en .F.voor secundaire indexen.
		SET UNIQUE	Heeft geen effect.
		APPEND, EDIT, BROWSE APPEND BLANK, REPLACE	Toont een waarschuwing (APPEND, EDIT, BROWSE) of een foutmelding (APPEND BLANK, REPLACE) en voegt geen records toe als dit een dubbele waarde oplevert. Dit geldt ook voor lege records. Een tabel kan niet meer dan één leeg record bevatten als een primaire of een unieke index actief is..



Windows-omgeving

In dit deel komen aspecten van het programmeren voor de Windows-omgeving aan bod.

Dit deel bevat de volgende hoofdstukken:

- Hoofdstuk 24, "Afdrukken"
- Hoofdstuk 25, "Werken met DLL's en de Windows-API"
- Hoofdstuk 26, "Gegevens uitwisselen met behulp van DDE"

Hoofdstuk

24

Afdrukken

dBASE beschikt over een geïntegreerd rapportontwerpprogramma met uitgebreide afdrukfuncties. Rapportontwerp genereert een bestand waarmee u rapporten kunt afdrukken met het commando REPORT FORM. Rapportontwerp is de beste oplossing voor bijna al uw afdrukbehoeften.

In dit hoofdstuk worden de programmeertechnieken besproken waarmee u kunt afdrukken met de dBASE-taal. Lees dit hoofdstuk als u wilt weten hoe u afdrukt zonder Rapportontwerp te gebruiken.

Werken met afdrukfuncties

In dBASE voor Windows worden de afdrukmogelijkheden van de Windows-omgeving gecombineerd met de bestaande afdrukmogelijkheden van dBASE IV. Hierbij worden de *systeemgeheugenvariabelen* van dBASE IV, waarmee u de opmaak van afdrukuitvoer bepaalt, ondersteund. Deze variabelen besturen alle aspecten van het afdrukken, van de selectie van het printeraansturingsprogramma tot de instelling van marges, regelafstand en fonts.

Nieuwe afdrukfuncties in dBASE voor Windows

- U kunt nu TrueType-fonts opgeven bij het commando ? voor dBASE.
- U kunt nu afdrukopties wijzigen vanuit programma's en u kunt de gebruiker een dialoogvenster laten openen waarin deze printers selecteert met de functie CHOOSEPRINTER( ).
- U kunt nu decimale tekenspatiëring opgeven bij de variabele _lmargin, omdat elk teken een aparte cel beslaat met eigen attributen. De ondersteuning van proportionele fonts biedt ook automatische aanpassing van de regelafstand, zodat grote fonts op een regel niet worden overlapt door de volgende regel.

- U kunt nu vanuit programma's de afdrukrichting wijzigen met een nieuwe printerbesturingsvariabele, _porientation. De afdrukrichting kan alleen worden gewijzigd bij pagina-einden. U kunt de richting van een afbeelding of tabel bijvoorbeeld niet halverwege de pagina wijzigen. Wanneer u _porientation wijzigt, moet u ook de variabele _plength in de code aanpassen. Als u de functie CHOOSEPRINTER( ) gebruikt om de afdrukrichting van een printer te wijzigen, wordt de geheugenvariabele _plength automatisch aangepast.

Printeraansturingsprogramma's opgeven

Een printeraansturingsprogramma is een programma waarmee printercodes worden ingevoegd in dBASE-uitvoer. Deze codes geven instructies aan de printer om bijvoorbeeld in te springen, fonts te wijzigen, vet te gebruiken, enzovoort.

In de Windows-omgeving installeert u printeraansturingsprogramma's en stelt u verbindingen met netwerkprinters in met het Configuratiescherm of met Afdrukbeheer. Zie de Windows-documentatie voor meer informatie.

In Windows 3.1 kunt u meerdere printeraansturingsprogramma's toewijzen aan dezelfde printerpoort. Op deze manier kunt u verschillende printeraansturingsprogramma's toewijzen aan verschillende applicaties. Als u een ander printeraansturingsprogramma wilt kiezen uit de geïnstalleerde printeraansturingsprogramma's, gebruikt u de variabele _pdriver vanuit een programma of selecteert u de printer met de functie CHOOSEPRINTER( ).

Gebruik een tekenreeks met begrenzingstekens voor de bestandsnaam van het printeraansturingsprogramma, een komma en de naam van de printer. In het volgende voorbeeld ziet u een toewijzingsopdracht waarmee het aansturingsprogramma voor een Postscript-printer wordt toegewezen aan _pdriver:

```
_pdriver = "PSCRIPT, HP LaserJet IIISi Postscript"
```

Hierbij is "PSCRIPT" de bestandsnaam van het printeraansturingsprogramma (de extensie .DRV wordt gebruikt als standaardoptie) en is "HP LaserJet IIISi Postscript" de naam en het type van de printer. U moet de precieze naam van de printer opgeven, omdat een printeraansturingsprogramma van Windows meerdere typen printers kan ondersteunen.

Opmerking Bij Postscript-printers is vooral de aanduiding van het Postscript-niveau (II of III) van belang, niet de merknaam van de printer.

Naam van printeraansturingsprogramma bepalen

U bepaalt als volgt de naam die in dBASE wordt gebruikt voor een printeraansturingsprogramma dat is geïnstalleerd in Windows 3.1. U kunt deze naam vervolgens gebruiken in uw applicatie.

1 Selecteer de gewenste printer als standaardprinter in het Configuratiescherm van Windows of met de functie CHOOSEPRINTER( ) in dBASE.

2 Vraag de dBASE-naam voor de printer op vanuit het commandovenster. Typ het volgende:

```
? _pdriver
```

De naam die wordt gebruikt voor het printeraansturingsprogramma in dBASE, wordt weergegeven in het resultatenpaneel van het commandovenster.

Doorlopende en niet-doorlopende uitvoer

Veel dBASE-uitvoercommando's zoals ?, ??, DIR, DISPLAY en LIST produceren *doorlopende uitvoer* (streaming output), terwijl commando's zoals EJECT en @...SAY *niet-doorlopende uitvoer* (non-streaming output) produceren. Deze typen uitvoer worden op verschillende manieren verwerkt in uw programma's en op verschillende manieren verzonden naar de printer.

De verschillen tussen doorlopende uitvoer en niet-doorlopende uitvoer zijn als volgt:

- Doorlopende uitvoer begint bij de huidige afdrukpositie, terwijl niet-doorlopende uitvoer wordt verzonden naar een specifieke lokatie op het scherm en de afgedrukte pagina. Als de huidige afdrukpositie bijvoorbeeld rij 5, kolom 7 is, wordt doorlopende uitvoer standaard verzonden naar deze positie. Niet-doorlopende uitvoer gaat naar de coördinaten die expliciet zijn opgegeven in het uitvoercommando.
- Bij doorlopende uitvoer volgt het ene teken op het andere. De plaatsing van elk teken is daarom meestal afhankelijk van de plaatsing van het vorige teken. Bij niet-doorlopende uitvoer is de plaatsing van een teken niet per se afhankelijk van de plaatsing van het vorige teken.
- Doorlopende uitvoer gaat naar alle beschikbare uitvoerapparaten, terwijl niet-doorlopende uitvoer alleen wordt verzonden naar het opgegeven apparaat. Als u doorlopende uitvoer bijvoorbeeld verzendt naar de printer, gaat de uitvoer naar het scherm en de printer. Als u niet-doorlopende uitvoer echter verzendt naar de printer, gaat de uitvoer alleen naar de printer.
- Doorlopende uitvoer kan worden opgemaakt met systeemgeheugenvariabelen, terwijl het uiterlijk van niet-doorlopende uitvoer wordt bepaald door het standaardfont en de standaardtekenstijl van uw printer.

Doorlopende uitvoer heeft de voorkeur boven niet-doorlopende uitvoer, omdat u het uiterlijk van doorlopende uitvoer kunt besturen met fonts, tekenstijlen en kolomopmaak. Niet-doorlopende uitvoer kan echter nuttig zijn wanneer u uitvoer verzendt naar invoerformulieren waarin tekens moeten worden ingevoegd op precieze lokaties en zonder tekstopmaak.

Uitvoer verzenden

dBASE kent een aantal taalelementen en bijbehorende menuhulpmiddelen waarmee u kunt instellen welke printer de uitvoer ontvangt. Hoe u de uitvoer verzendt, is afhankelijk van het type uitvoer (doorlopend of niet-doorlopend) dat u verzendt.

In Windows drukt u *document-georiënteerd* af, dat wil zeggen dat elke afdrukbewerking wordt behandeld als het openen of sluiten van een document. Dit document wordt meestal *afdrukbestand* genoemd. U moet een afdrukbestand openen voordat u uitvoer kunt genereren, en sluiten voordat u de uitvoer kunt verzenden naar de printer.

Doorlopende uitvoer verzenden

U verzend doorlopende uitvoer door afdrukbestanden te openen en sluiten. U kunt dit doen op de volgende twee manieren:

- SET PRINTER TO <*apparaat*>. Hiermee stelt u de doelprinter in en opent u een afdrukbestand voor deze printer. Als u al eerder een afdrukbestand hebt geopend, wordt dit bestand gesloten met SET PRINTER TO <*apparaat*> voordat het nieuwe afdrukbestand wordt geopend.
- CHOOSEPRINTER( ). Dit is een interactieve methode om printers te selecteren in het dialoogvenster **Printerinstellingen**. Wanneer u een nieuwe printer selecteert en **OK** kiest, worden alle relevante systeemgeheugenvariabelen opnieuw ingesteld. Daarnaast wordt impliciet het commando SET PRINTER TO <*apparaat*> opgegeven. Wanneer u **OK** kiest om de nieuwe instelling te accepteren, wordt het dialoogvenster **Printerinstellingen** gesloten en resulteert CHOOSEPRINTER( ) in .T. (waar). CHOOSEPRINTER( ) stelt vervolgens _pdriver opnieuw in.

Voordat u uitvoer kunt verzenden, moet u een afdrukbestand openen met SET PRINTER TO <*apparaat*> of CHOOSEPRINTER( ). U moet het afdrukbestand sluiten met CLOSE PRINTER of het afdrukbestand sluiten en een nieuw afdrukbestand maken met SET PRINTER TO <*apparaat*>, om de uitvoer af te drukken nadat u deze hebt verzonden. U doet dit als volgt:

1 Open het afdrukbestand met SET PRINTER TO <*apparaat*> of CHOOSEPRINTER( ) om uitvoer te verzenden.

   U kunt de uitvoer ook verzenden naar een schijfbestand met SET PRINTER TO FILE <*bestandsnaam*>. De uitvoer wordt afgedrukt naar een schijfbestand, waarin de uitvoer en printerstuurcodes voor de opgegeven printer worden opgeslagen.

2 Verzend de uitvoer naar de printer met SET PRINTER ON.

3 Genereer de uitvoer met commando's zoals ?, ??, DIR, DISPLAY en LIST.

4 Stop de verzending van uitvoer naar de printer met SET PRINTER OFF.

5 Sluit het huidige afdrukdocument met CLOSE PRINTER of SET PRINTER TO.

In het volgende voorbeeld opent u een afdrukbestand, verzendt u uitvoer en sluit u het bestand, met SET PRINTER TO:

```
SET PRINTER TO LPT1  && Uitvoerapparaat opgeven
SET PRINTER ON       && Uitvoer verzenden naar printer
DIR *.*              && Directorylijst weergeven voor uitvoerapparaat
SET PRINTER OFF      && Geen uitvoer meer verzenden naar printer
CLOSE PRINTER        && Afdrukbestand sluiten
```

In het volgende voorbeeld opent u een afdrukdocument in het dialoogvenster **Printerinstellingen**, verzendt u de uitvoer en sluit u het bestand, met CHOOSEPRINTER( ):

```
CHOOSEPRINTER()   && Printer selecteren
SET PRINTER ON    && Uitvoer verzenden naar printer
DIR *.*           && Directorylijst weergeven voor uitvoerapparaat
SET PRINTER OFF   && Directorylijst weergeven voor uitvoerapparaat
CLOSE PRINTER     && Afdrukbestand sluiten
```

Niet-doorlopende uitvoer verzenden

Net als bij doorlopende uitvoer verzendt u niet-doorlopende uitvoer met SET PRINTER TO *<apparaat>*. U moet echter ook SET DEVICE TO PRINTER instellen om de uitvoer te verzenden naar de printer in plaats van het scherm. Als u geen apparaat opgeeft bij SET PRINTER TO, wordt niet-doorlopende uitvoer naar de printer gezonden door SET DEVICE TO PRINTER. U kunt de uitvoer ook verzenden naar een schijfbestand met SET DEVICE TO FILE *<bestandsnaam>*.

In het volgende voorbeeld verzendt u uitvoer naar de printer die is verbonden met parallelle poort LPT2 via Afdrukbeheer.

```
SET PRINTER TO LPT2                          && Uitvoerapparaat opgeven
SET DEVICE TO PRINTER                        && Uitvoer verzenden naar printer
@ 12, 1 SAY "Dit begint op rij 12, kolom 1"  && Bevestigingsmelding
SET DEVICE TO SCREEN                         && Uitvoer doorsturen naar scherm
CLOSE PRINTER                                && Afdrukbestand sluiten
```

In tegenstelling tot SET PRINTER ON/OFF wordt de uitvoer niet weergegeven op het scherm, omdat de uitvoer naar slechts één apparaat wordt verzonden met SET DEVICE.

Printerbeschikbaarheid testen

U kunt de volgende fouten opsporen met de functie PRINTSTATUS( ):

- Er is niet voldoende schijfruimte of geheugen om het uitvoerbestand te verzenden naar de spooler
- De printer is verbonden met een andere poort dan de poort die u hebt opgegeven bij SET PRINTER TO of CHOOSEPRINTER( )

Wanneer een van deze fouten optreedt, resulteert PRINTSTATUS( ) in .F. (onwaar). Andere fouten (bijvoorbeeld geen papier) worden rechtstreeks afgehandeld door Afdrukbeheer.

Met PRINTSTATUS( ) kunt u potentiële printerproblemen ontdekken voordat deze een fout veroorzaken voor Afdrukbeheer. In het volgende voorbeeld test u bijvoorbeeld de afdrukstatus en geeft u een waarschuwing weer als de afdrukstatus niet OK is:

```
IF .NOT. PRINTSTATUS()                    && Printerbeschikbaarheid controleren
   WAIT "Let op, probleem met de printer."  && Waarschuwing
   RETURN
ENDIF
```

Uitvoer afdrukken naar een bestand

Druk uitvoer af naar een bestand wanneer u de uitvoer wilt afdrukken rechtstreeks vanuit DOS, of wanneer u een bestand wilt maken dat kan worden afgedrukt zonder het programma te gebruiken waarmee het bestand is gemaakt.

In Windows wordt uitvoer meestal in een tijdelijk bestand op de schijf geplaatst voordat de uitvoer wordt verzonden naar de printer. De uitvoer wordt verzonden naar de printer vanuit dit bestand, waarna het bestand wordt verwijderd. Als u SET PRINTER TO FILE <*bestandsnaam*> of SET DEVICE TO FILE <*bestandsnaam*> gebruikt, wordt er geen uitvoer verzonden naar de printer en wordt de uitvoer geschreven naar het opgegeven schijfbestand. Omdat dit bestand printerstuurcodes bevat, is het bestand niet altijd een leesbaar standaardtekstbestand. U kunt het bestand dan afdrukken met de opdracht COPY <*bestandsnaam*> LPT1 in DOS of Windows, zonder dBASE te gebruiken.

Uitvoertekens plaatsen

Wanneer u uitvoer rechtstreeks verzendt naar de printer, wordt elk teken geplaatst op het coördinatenvlak, een denkbeeldig tweedimensionaal raster waarop tekens worden ingedeeld. De breedte van de tekencellen is afhankelijk van de waarde van _ppitch, terwijl de hoogte van elke tekencel altijd 1/6 inch is. Wanneer de lettergrootte van het huidige font groter is dan deze hoogte, neemt elke uitvoerregel meer dan een rij met tekencellen in beslag. (Zie *Commando's en functies*, _ppitch, voor een tabel met mogelijke waarden voor _ppitch.)

Uitvoer afdrukken op absolute coördinaten

Met het commando @...SAY verzendt u niet-doorlopende uitvoer naar specifieke lokaties op het scherm of op de afgedrukte pagina met *absolute adressering*. Bij absolute adressering is de plaatsing van een teken niet afhankelijk van de plaatsing van het vorige teken.

In het volgende voorbeeld verzendt u uitvoer naar specifieke regel- en kolomlokaties op de afgedrukte pagina. De printer moet boven aan de pagina staan, omdat u niet kunt teruggaan naar een vorige rij bij afdrukuitvoer. Het is daarom gemakkelijker relatieve coördinaten te gebruiken voor afdrukuitvoer.

```
SET DEVICE TO PRINTER
@ 2, 1 SAY "Dit begint op rij 2, kolom 1"
@ 3, 10 SAY "Dit begint op rij 3, kolom 10"
CLOSE PRINTER
SET DEVICE TO SCREEN
```

De afdrukuitvoer ziet er als volgt uit:

```
Dit begint op rij 2, kolom 1
         Dit begint op rij 3, kolom 10
```

Uitvoer afdrukken op relatieve coördinaten

Wanneer u niet-doorlopende uitvoer afdrukt, kunt u de afdruk soms het beste relatief plaatsen ten opzichte van de huidige afdrukpositie van de printer. Bij rapporten die uitvoer afdrukken met variabele lengten, moet bijvoorbeeld de positie voor elke nieuwe gegevensstroom worden berekend ten opzichte van een vorige gegevensstroom.

Met de functies PROW( ) en PCOL( ) plaatst u uitvoer relatief ten opzichte van de huidige afdrukpositie. PROW( ) resulteert in het rijnummer en PCOL( ) resulteert in het kolomnummer waarop de printer wordt ingesteld om te beginnen met afdrukken.

Omdat u PROW ( ) en PCOL( ) kunt gebruiken om de huidige positie van de printer te bepalen, kunt u deze functies ook gebruiken om de coördinaten van andere posities te berekenen ten opzichte van de huidige afdrukpositie. Dit wordt *relatieve adressering* genoemd. In het volgende voorbeeld drukt u bijvoorbeeld een tekenreeks af op vijf rijen onder en vier kolommen naar rechts ten opzichte van de huidige afdrukpositie op het coördinatenvlak:

```
SET DEVICE TO PRINTER
@ PROW() + 5, PCOL() + 4 SAY "Een voorbeeld van relatieve adressering"
SET DEVICE TO SCREEN
CLOSE PRINTER
```

Decimale coördinaten

U kunt decimale of gehele waarden opgeven bij AT <*Nuitdr*>, een optie waarmee u de horizontale plaatsing van het eerste teken op een regel instelt.

Het effect van de waarde die u opgeeft bij AT, is afhankelijk van de huidige instelling van _ppitch. Als u _ppitch bijvoorbeeld instelt op "ELITE," is de tekendichtheid 12 tekens per inch. Als u bijvoorbeeld 1,85 inch (4,7 cm) wilt inspringen, vermenigvuldigt u 1,85 met 12. Het resultaat (22,2) geeft u op bij AT, bijvoorbeeld:

```
SET PRINTER ON
_ppitch = "ELITE"                                    && 12 tekens per inch.

? "Deze reeks begint op 1,85 inch vanaf de linkerkant." AT 22.2  && 12 * 1,85 = 22,2
SET PRINTER OFF
CLOSE PRINTER
```

Verticale en horizontale afdrukpositie instellen

U stelt de verticale afdrukpositie van de printer (de afdrukuitvoerrij) opnieuw in met SET PROW en de horizontale afdrukpositie (de afdrukuitvoerkolom) met SET PCOL. De coördinaten van een volgend commando @...SAY worden geëvalueerd ten opzichte van de nieuwe waarden voor PROW( ) en PCOL( ).

Als de afdrukpositie bijvoorbeeld op rij 10 staat wanneer u SET PROW TO instelt op 0, wordt de melding "Welke regel?" in het volgende voorbeeld afgedrukt vanaf rij 7 (dit was rij 17 vóór het commando SET PROW):

```
@ 7, 20 SAY "Welke regel?"
```

Regelnummers, papierdoorvoer en aantal exemplaren instellen

Een afdruktaak is een methode waarmee u regelnummers kunt instellen of bepalen, automatische papierdoorvoer kunt uitvoeren of meer dan een exemplaar van uw uitvoer kunt afdrukken. U stelt deze mogelijkheden in met de volgende drie systeemgeheugenvariabelen:

- _pcopies. Hiermee stelt u in hoeveel exemplaren u wilt afdrukken wanneer een afdruktaak wordt uitgevoerd.
- _peject. Hiermee stelt u in of en wanneer de printer een blad papier moet doorvoeren.
- _plineno. Hiermee stelt u het huidige regelnummer voor de afdrukuitvoer in.

Deze variabelen zijn alleen actief tijdens een afdruktaak. Wanneer u uitvoer genereert op een andere manier, worden de variabelen genegeerd. Als u een afdruktaak wilt genereren met programmacode, begint u de sectie met PRINTJOB (hiermee opent u de afdruktaak) en sluit u de sectie af met ENDPRINTJOB (hiermee sluit u de afdruktaak). De variabelen _pcopies, _peject en _plineno zijn dan actief tijdens dit gedeelte van het programma.

In het volgende voorbeeld ziet u een programma waarmee een afdruktaak wordt uitgevoerd:

```
SET TALK OFF
USE AFNEMERS
_peject = "AFTER"
_pcopies = 2
SET PRINTER ON                                  && Uitvoer verzenden naar printer
PRINTJOB                                        && Printtaak openen
   * Datum en tijd afdrukken boven aan elke
   * opeenvolgende pagina (behalve de eerste)
   ON PAGE AT LINE _plength - 1 DO PagEinde     && Regelnummer instellen
                                                && voor pagina-einde
   SCAN                                         && Tabel doorzoeken
      ?? Afnemernr, Bedrijf, Contact            && Velden afdrukken
      ?                                         && Lege regel invoegen
   ENDSCAN
ENDPRINTJOB                                     && Afdruktaak sluiten
SET PRINTER OFF                                 && Geen uitvoer meer verzenden
                                                && naar printer
CLOSE PRINTER                                   && Afdrukbestand sluiten
RETURN

PROCEDURE PagEinde
EJECT
* _pageno automatisch verhogen
?? DATE() AT 1, "Pagina:" AT 67, _pageno PICTURE "999" AT 73  && Paginakoptekst afdrukken
?
?
RETURN
```

Bij ON PAGE geeft u een commando op dat u wilt uitvoeren wanneer de afdrukuitvoer een bepaald regelnummer bereikt. In dit voorbeeld bepaalt ON PAGE wanneer een

pagina-einde wordt ingevoegd. Telkens wanneer het huidige regelnummer het einde van een pagina bereikt, wordt het papier doorgevoerd en worden de datum en het paginanummer afgedrukt boven aan de volgende pagina (procedure PagEinde). Er worden twee exemplaren van de afdruktaak afgedrukt (_pcopies = 2) en er wordt een lege pagina doorgevoerd nadat de laatste pagina is afgedrukt (_peject = "AFTER").

Afdrukuitvoer opmaken

U kunt uw afgedrukte document opmaken met geheugenvariabelen, om het document een verzorgd en professioneel uiterlijk te geven.

U kunt systeemgeheugenvariabelen wijzigen in het commandovenster en vanuit programma's. Met LIST MEMORY geeft u de huidige waarden van alle systeemgeheugenvariabelen weer. U kunt bepaalde printerinstellingen interactief wijzigen met het dialoogvenster **Printerinstellingen**. Kies **Bestand | Printerinstellingen** om het dialoogvenster weer te geven. In dit dialoogvenster kunt u de standaardprinter, het papierformaat en de afdrukrichting wijzigen. Elke wijziging van een systeemgeheugenvariabele vervangt de vorige instelling, ongeacht of de variabele is gewijzigd vanuit een programma of in het dialoogvenster **Printerinstellingen**.

U kunt uw uitvoer ook opmaken met de opties AT, FUNCTION en STYLE van commando's voor doorlopende uitvoer.

Paginalengte aanpassen

Met de variabele _plength stelt u het aantal regels per pagina in. Als de standaardinstelling van de printer voor het aantal regels per pagina bijvoorbeeld 6 regels per inch is, maar u 8 regels wilt afdrukken in de gecomprimeerde modus, moet u _plength verhogen van 66 naar 88 (voor 11 inch-papier).

U kunt een geheel of decimaal getal opgeven bij _plength, zodat u de paginalengte kunt aanpassen aan proportionele fonts of documenten met ongebruikelijke afmetingen (bijvoorbeeld salarischeques met een regelafstand die afwijkt van het huidige font). Als u deze salarischeques afdrukt, kunt u _plength gebruiken om de regelafstand aan te passen.

Als u een matrixprinter gebruikt die rechtstreeks is verbonden met de computer waarop u afdrukt, kunt u een afdrukbewerking stoppen en het papier handmatig verticaal verschuiven. U kunt deze wijziging dan compenseren met de systeemgeheugenvariabele _plineno voordat u doorgaat met afdrukken. Nadat u het papier hebt verschoven, stelt u de variabele _plineno in op het regelnummer waarop de schrijfkop nu staat. Vervolgens gaat u door met afdrukken. U kunt een geheel of decimaal getal opgeven bij _plineno, zodat u de positie van de schrijfkop zeer nauwkeurig kunt instellen.

Staande of liggende richting opgeven

Als uw printer deze mogelijkheid ondersteunt, kunt u afdrukken in staande of liggende richting met de systeemgeheugenvariabele _porientation. Stel de staande richting in

door _porientation="portrait" op te geven of de liggende richting door _porientation="landscape" op te geven. Vergeet niet de variabele _plength te wijzigen van 55 of 60 regels (voor papier van 11 inches) in 51 regels voor een volledige pagina of 45 regels voor een pagina met een boven- en ondermarge van 3 regels (voor 8,5 inch-papier).

Gebruikers afdrukopties laten selecteren

U kunt de gebruiker van uw applicatie de printerinstellingen laten wijzigen in het dialoogvenster **Printerinstellingen**. Neem de volgende code op om het dialoogvenster **Printerinstellingen** weer te geven met de functie CHOOSEPRINTER( ):

```
? CHOOSEPRINTER()
```

Als de gebruiker wijzigingen aanbrengt in dit dialoogvenster en **OK** kiest om de wijzigingen te accepteren, resulteert CHOOSEPRINTER( ) in .T. (waar). De geselecteerde waarden worden geëvalueerd en de systeemgeheugenvariabelen worden opnieuw ingesteld. Als de gebruiker **Annuleren** kiest, resulteert de functie in .F. (onwaar) en blijven de systeemgeheugenvariabelen ongewijzigd.

De wijzigingen die de gebruiker aanbrengt in het dialoogvenster **Printerinstellingen**, beïnvloeden de instelling van SET PRINTER TO en wijzigen de inhoud van de drie systeemgeheugenvariabelen _pdriver, _plength en _porientation.

Marges en inspringing opgeven

Er zijn diverse systeemgeheugenvariabelen waarmee u de marges en inspringing van afdrukuitvoer kunt instellen. Bij deze variabelen geeft u numerieke waarden op om de breedte (in kolommen) aan te geven van de marge of inspringing. U kunt gehele of decimale getallen opgeven, zodat u marges en inspringingen zeer nauwkeurig kunt instellen.

Tabel 24.1 Systeemgeheugenvariabelen voor marges en inspringing

Variabele	Doel
_ploffset	Hiermee stelt u de afstand in vanaf de linkerrand van het papier tot de linkermarge van het *afdrukgebied*. De variabele is meestal ingesteld op 0. U kunt deze variabele gebruiken om rekening te houden met de breedte van gaatjes in geperforeerd papier.
_lmargin	Hiermee stelt u de linkermarge van afdrukuitvoer in vanaf het begin van het afdrukgebied.
_rmargin	Hiermee stelt u de rechtermarge van afdrukuitvoer in vanaf het einde van het afdrukgebied.
_pcolno	Hiermee stelt u de kolom in waarop de eerste regel van een uitvoerstroom wordt afgedrukt.
_indent	Hiermee stelt u een extra inspringing in voor de eerste regel van een alinea.

De opgegeven waarden zijn cumulatief. De totale afstand vanaf de linkerrand van het papier tot het eerste teken in een alinea is bijvoorbeeld de waarde van _ploffset plus het aantal kolommen dat u opgeeft bij _indent en _lmargin.

Horizontale uitlijning opgeven

Met de variabele _alignment kunt u uitvoer links of rechts uitlijnen of centreren binnen de marges die u instelt met _lmargin en _rmargin, of met _ploffset en _rmargin. Als u _alignment, _lmargin of _rmargin wilt gebruiken, moet u _wrap instellen op .T. (waar).

Regelafstand en tabstops aanpassen

Met de variabele _pspacing stelt u de regelafstand in. Als u _pspacing bijvoorbeeld instelt op 2, wordt de afdrukuitvoer afgedrukt met dubbele regelafstand, en als u de variabele instelt op 3, wordt de uitvoer afgedrukt met driedubbele regelafstand. U kunt gehele of decimale getallen opgeven, zodat u de verticale afstand zeer nauwkeurig kunt instellen, bijvoorbeeld:

```
_pspacing = 2
_pspacing = 1.5
```

Met de variabele _tabs stelt u de lengte van tabstops in in aantallen tekens. U kunt alleen gehele getallen opgeven (die u tussen begrenzingstekens moet zetten), bijvoorbeeld:

```
_tabs = "5,10,15,20"
```

Als u geen tabs instelt, wordt de DOS-standaardwaarde van één tab per 8 spaties gebruikt.

Paginadoorvoer en paginering

U kunt paginadoorvoer en paginering instellen wanneer u meerdere pagina's uitvoer verzendt naar de printer.

Paginadoorvoer vs. regeldoorvoer

Met de variabele _padvance kunt u instellen of u het papier wilt doorvoeren in de printer met paginadoorvoer of regeldoorvoer.

Als u de standaardinstelling "formfeed" (paginadoorvoer) opgeeft, wordt het papier per pagina doorgevoerd, waarbij de printer naar de bovenkant van de volgende pagina gaat op basis van de standaardinstelling van de printer voor de paginalengte.

Voor een matrixprinter kunt u een kortere paginalengte opgeven met de instelling "linefeeds" (regeldoorvoer). U kunt dan korte pagina's afdrukken, zoals cheques met een lengte van 20 regels. Stel _plength in op de lengte van de uitvoer (20 in dit voorbeeld) en stelt _padvance in op "linefeeds".

Als u in dit voorbeeld cheques wilt afdrukken op een laserprinter, moet u een pagina samenstellen met drie cheques en paginadoorvoer opgeven om het papier door te voeren.

Paginanummers en paginabereik

Wanneer u meerdere pagina's uitvoer genereert, kunt u instellen welke pagina's u wilt afdrukken met de variabelen _pageno, _pbpage en _pepage. Met _pageno stelt u het huidige paginanummer in, met _pbpage_ stelt u de eerste pagina van een paginabereik

in en met _pepage stelt u de laatste pagina van een paginabereik in. Pagina's met een waarde voor _pageno die kleiner is dan de waarde voor _pbpage of groter is dan de waarde voor _pepage, worden niet afgedrukt.

Tijdens het genereren van de uitvoer wordt het huidige paginanummer (_pageno) voortdurend gecontroleerd en worden pagina's alleen afgedrukt wanneer _pageno groter is dan of gelijk is aan _pbpage. Als _pageno kleiner is dan _pbpage of groter is dan _pepage, wordt de uitvoer intern verschoven, maar niet afgedrukt.

U kunt het paginanummer van de huidige pagina op elk moment wijzigen door de waarde voor _pageno te wijzigen.

Kwaliteitmodus vs. kladmodus

De meeste matrixprinters kunnen afdrukken in kwaliteitmodus of kladmodus. Met de kwaliteitmodus wordt de uitvoer afgedrukt met een hogere kwaliteit (fijnere resolutie) dan met de kladmodus. De uitvoer wordt echter vaak sneller afgedrukt in kladmodus. U stelt kwaliteit- of kladmodus in met de variabele _pquality. Wanneer _pquality .T. (waar) is, wordt de uitvoer afgedrukt in kwaliteitmodus. Als _pquality .F. (onwaar) is, wordt de uitvoer afgedrukt in kladmodus.

Omdat laserprinters geen onderscheid maken tussen kwaliteitmodus en kladmodus, verschijnt een foutmelding wanneer u _pquality instelt op .T.

Fonts en tekststijlen opgeven

Het uiterlijk van doorlopende uitvoer is afhankelijk van het font en de tekststijl die u selecteert. Een *font* (ook wel *lettertype* genoemd) is een verzameling letters, cijfers, symbolen en leestekens die een gemeenschappelijke vorm of uiterlijk hebben. Een *tekststijl* (ook wel *fontstijl*) is een attribuut, zoals vet, cursief of onderstreept. In dBASE kunt u fonts gebruiken die zijn geïnstalleerd in het Configuratiescherm van Windows.

De tekststijlen die u kunt gebruiken, zijn afhankelijk van het font dat u selecteert. Bij sommige fonts kunt u slechts een paar (of geen) tekststijlen gebruiken, terwijl andere fonts een groot aantal tekststijlen toestaan. Wanneer u doorlopende uitvoer verzendt naar de printer met ? of ??, kunt u het font en de tekststijl instellen met de optie STYLE.

Windows-fonts gebruiken in dBASE

Voordat u een font kunt gebruiken in dBASE, moet u een identificatienummer toewijzen aan het font. Bewerk de sectie [Fonts] in het bestand DBASEWIN.INI om een identificatienummer toe te wijzen aan de fonts die u wilt gebruiken. Raadpleeg Appendix C in het *Handboek* voor meer informatie over de instellingen van DBASEWIN.INI. De sectie [Fonts] ziet eruit als volgt:

```
[Fonts]
1=Times New Roman, 12, ROMAN
2=Arial, 10, SWISS
```

Beide instellingen in dit voorbeeld bestaan uit de volgende vier onderdelen:

- De numerieke identificatie (1 en 2). Dit is het nummer dat het font aangeeft in de STYLE-uitdrukking van commando's voor doorlopende uitvoer zoals ? of ??. Het identificatienummer kan een getal zijn tussen 1 en 32.767. In de praktijk installeert u waarschijnlijk niet meer dan enkele tientallen fonts. U kunt de fonts dan groeperen per tiental, bijvoorbeeld Times Roman-fonts van 1 tot 10 (voor de diverse lettergrootten), Arial-fonts van 11 tot 20, enzovoort. Met een dergelijke conventie kunt u de fontgroepen gemakkelijk onthouden bij de codering van fonts.
- De fontnaam (in het voorbeeld Times New Roman en Arial). De fonts die u wilt gebruiken, moeten zijn geïnstalleerd in uw Windows-besturingssysteem. (Geef het dialoogvenster **Lettertypen** in het Configuratiescherm van Windows weer om te zien welke fonts er zijn geïnstalleerd.)
- De lettergrootte (12 en 10). Deze instelling bepaalt de hoogte en breedte van de tekens.
- De fontfamilie (ROMAN en SWISS). Een font kan lid zijn van een van de volgende families: ROMAN, SWISS, MODERN, SCRIPT of DECORATIVE. Als het opgegeven font niet beschikbaar is op het huidige uitvoerapparaat, wordt er automatisch een ander font van dezelfde fontfamilie gekozen.

Fonts

Geef bij een commando voor doorlopende uitvoer het nummer op dat u hebt toegewezen aan een font in DBASEWIN.INI, om de uitvoer af te drukken met dit font.

In het vorige voorbeeld zijn twee fonts opgegeven. Met het volgende commando genereert u de tekenreeks "Hallo" en gebruikt u het font Times New Roman met lettergrootte 12:

```
? "Hallo" STYLE 1
```

Met het volgende commando genereert u op dezelfde manier de tekenreeks "Tot ziens" en gebruikt u het font Arial met lettergrootte 10:

```
? "Tot ziens" STYLE 2
```

Tekststijlen

De optie STYLE van dBASE biedt vijf tekststijlen (zie Tabel 24.2). Deze stijlen zijn beschikbaar voor alle geïnstalleerde Windows-fonts.

- B. Hiermee geeft u vet als tekststijl op
- I. Hiermee geeft u cursief als tekststijl op
- U. Hiermee geeft u onderstreept als tekststijl op
- R. Hiermee geeft u hogere tekststijl (superscript) op
- L. Hiermee geeft u lagere tekststijl (subscript) op

Geef de stijlletter bij de optie STYLE op om een tekststijl te gebruiken. U kunt tekststijlen ook combineren.

Tabel 24.2 Voorbeelden van tekststijlen

Commando	Resultaat
`? "Dit is een vette zin" STYLE "B"`	**Dit is een vette zin**
`? "Dit is een cursieve zin" STYLE "I"`	*Dit is een cursieve zin*
`? "Font Times New Roman, onderstreept"; STYLE "1U"`	Font Times New Roman, onderstreept
`? "Font Arial, lettergrootte 10 en cursief"; STYLE "2I"`	*Font Arial, lettergrootte 10 en cursief*

Regelovergang instellen

U kunt een lange tekenreeks vaak het beste laten 'doorlopen' door de reeks te verdelen over twee of meer regels en verticaal te stapelen. Op deze manier kunt u bijvoorbeeld voorkomen dat een reeks bepaalde gedeelten van een uitvoerpagina overlapt. Dit proces wordt regelovergang of *verticaal rekken* genoemd.

U kunt de breedte van de doorlopende regels opgeven in gehele of decimale getallen.

Regelbreedte voor verticaal uitbreiden opgeven

U stelt verticaal rekken in met het functiesymbool V. U geeft de breedte van het veld op in tekens. Als u de breedte wilt berekenen in inches, moet u rekening houden met de pitch van de tekens. Bij doorlopende uitvoer wordt pitch PICA, ELITE of CONDENSED gebruikt. PICA is 10 tekens per inch, ELITE is 12 tekens per inch en CONDENSED is meestal 17,16 tekens per inch. Dit betekent dat een marge van 2 inch 20 tekens in PICA, 24 tekens in ELITE en 34 tekens in CONDENSED beslaat. U kunt de pitch opgeven met de systeemvariabele _ppitch.

In het volgende voorbeeld drukt u een tekenreeks af met een doorloopbreedte van 2,88 inch (7,2 cm):

```
SET PRINTER ON
_ppitch = "ELITE"                    && Tekendichtheid van 12 tekens per inch

? "Dit is een zeer lange reeks die doorloopt binnen de kolom.";
   FUNCTION "V34.56"                 && 2.88 * 12 = 34,56
SET PRINTER OFF
CLOSE PRINTER
```

Horizontale uitlijning voor verticaal uitbreiden opgeven

U kunt de horizontale uitlijning opgeven in gehele en decimale getallen.

U kunt de afdrukuitvoer van variabele pitch-instellingen uitlijnen met de functiesymbolen I en J. Met functiesymbool I centreert u elke doorlopende regel ten

opzichte van de andere regels en met functiesymbool J lijnt u elke doorlopende regel rechts uit.

In het volgende voorbeeld drukt u een tekenreeks af met een doorloopbreedte van 2,88 inches en centreert u alle doorlopende regels ten opzichte van elkaar:

```
SET PRINTER ON
_ppitch = "ELITE"          && Tekendichtheid van 12 tekens per inch

? "Dit is een zeer lange reeks die doorloopt binnen de kolom.";
FUNCTION "V34.56I"         && 2.88 * 12 = 34,56; "I" centreert doorlopende regels
SET PRINTER OFF
CLOSE PRINTER
```

Hoofdstuk

25

Werken met DLL's en de Windows-API

DLL's (Dynamic Link Library) zijn bestanden met gecompileerde programmacode die door Windows-toepassingen tijdens de uitvoering kan worden geladen en uitgevoerd. Deze programmacode kan subroutines en resources bevatten, zoals bitmaps en pictogrammen. Met DLL's kunt u niet-dBASE-functies aanroepen in uw dBASE-code.

De Windows-API (Application Programming Interface) is een ingebouwde bibliotheek in de Windows-omgeving met honderden C-functies die zijn opgeslagen in DLL's. Met de DLL-ondersteuning van dBASE kunnen programmeurs toegang krijgen tot de Windows-API.

In dit hoofdstuk wordt het concept van dynamische koppelingen behandeld en wordt beschreven hoe u niet-dBASE-functies kunt aanroepen met DLL's en de Windows-API. In dBASE wordt de toegang tot DLL gemakkelijk gemaakt voor programmeurs die gewend zijn externe functies aan te roepen. De aanroep van DLL-functies is echter een geavanceerde techniek waarbij kennis van andere programmeertalen, met name van de gebruikte gegevenstypen, goed van pas komt.

Werken met bibliotheken en koppelingen

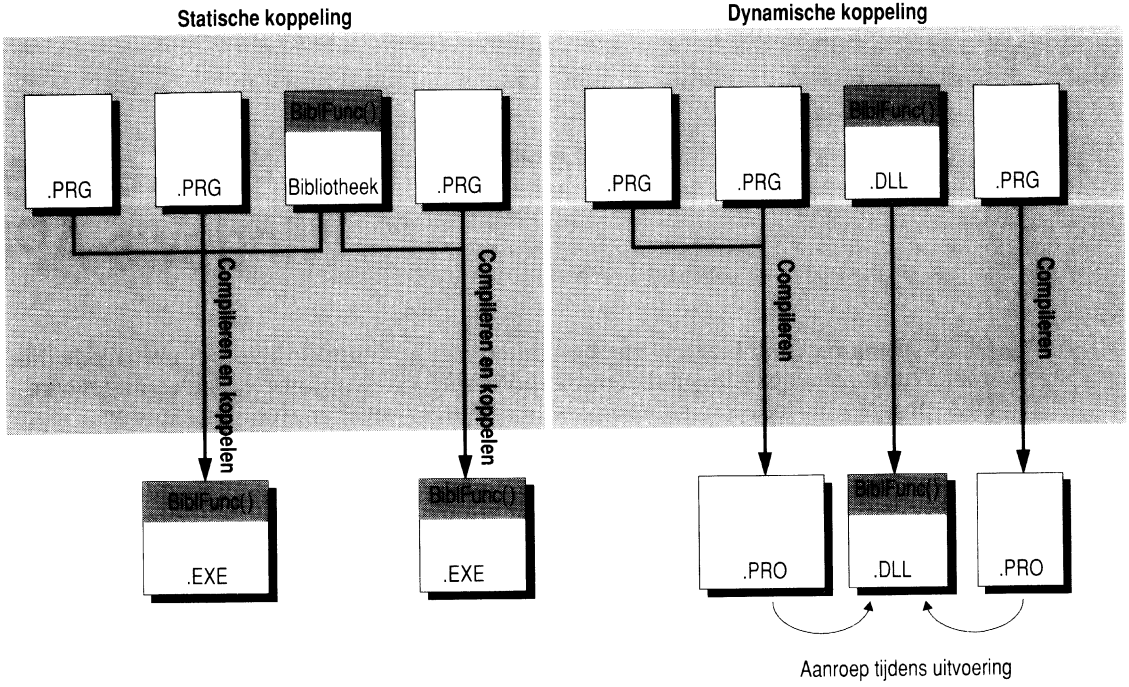
De termen "bibliotheek" en "koppeling" hebben verschillende betekenissen in verschillende programmatalen. In dBASE III PLUS onderhouden programmeurs bibliotheken van veel gebruikte routines en gebruiken ze deze routines in applicaties zonder te koppelen. In dBASE IV gebruiken ontwikkelaars DBLINK om gecompileerde programma's en bibliotheekroutines te koppelen tot één groot objectbestand.

Ontwikkelaars die de dBASE Compiler for DOS gebruiken, brengen een statische koppeling tot stand wanneer ze een uitvoerbaar bestand maken op basis van diverse objectmodulen en runtime-bibliotheken. Statische koppelingen treden alleen in werking op het moment dat u een applicatie maakt en zijn dus eenmaal actief. In Windows-

toepassingen worden daarentegen dynamische koppelingen tot stand gebracht door routines en resources aan te roepen tijdens uitvoering. Dynamische koppelingen treden in werking tijdens de uitvoering van een applicatie en kunnen dus vele malen plaatsvinden.

In Afbeelding 25.1 ziet u het verschil tussen statische koppeling (links) en dynamische koppeling (rechts).

Afbeelding 25.1 Vergelijking van statische koppeling en dynamische koppeling



De techniek waarbij gecompileerde routines worden onderhouden in aparte bestanden en deze bestanden tijdens uitvoering worden gekoppeld aan applicaties heeft de volgende voordelen:

- U kunt dezelfde bibliotheek gezamenlijk gebruiken in een groot aantal applicaties, in de 386 'enhanced'-modus van Windows zelfs in twee applicaties die gelijktijdig actief zijn
- U gebruikt minder geheugen-resources, omdat de bibliotheek slechts eenmaal wordt geladen in het geheugen
- U hoeft de bibliotheek niet telkens opnieuw te compileren of koppelen wanneer u uw applicatie samenstelt
- U maakt kleinere uitvoerbare bestanden

U kunt bijna elke functie aanroepen in een DLL, mits u de verwachte parameters, de resulterende waarde en de aanroepconventie van de functie kent.

De meeste programmabestanden van Windows-toepassingen en de Windows-besturingsomgeving zelf zijn programmamodulen (.EXE) of DLL's. Omdat DLL's onderdelen van toepassingen zijn, staan de DLL's al op de schijf wanneer een van de DLL-routines of -resources wordt aangeroepen in een programma.

DLL-bestanden kunnen diverse extensies hebben (bijvoorbeeld .DLL, .DRV, .FON of .EXE), hoewel de meeste DLL-bestanden de extensie .DLL hebben. De eerste acht tekens van een DLL die wordt geladen in het geheugen door een applicatie, moeten uniek zijn, ongeacht de extensie. Als u dus SCRIPT.FON hebt geladen, kunt u daarna niet SCRIPT.DLL laden. Het commando waarmee u SCRIPT.DLL probeert te laden (EXTERN of LOAD DLL), wordt niet uitgevoerd en er wordt geen fout gemeld.

DLL's gebruiken in dBASE

DLL's vormen een brug naar C, Pascal, assembleertalen en andere niet-dBASE-functies. Programmeurs die ervaring hebben met een andere taal, kunnen eigen functies schrijven en deze compileren met een Windows-compiler. Bovendien brengen veel externe leveranciers bibliotheken met functies voor Windows-toepassingen op de markt. Omdat Windows is geschreven in C, zijn de meeste DLL-functies geschreven in C of C++.

Leveranciers van DLL's moeten informatie geven over het gebruik van de bibliotheekfuncties. U moet het volgende weten om een functie in een DLL te gebruiken:

- De naam van de functie.
- De naam van het DLL-bestand met de functie.
- De gegevenstypen van de parameters en de resulterende waarde (indien van toepassing).
- De aanroepconventie. De meeste Windows-functies maken gebruik van de Pascal-aanroepconventie, waarbij een vast aantal parameters is vereist. Bij de C-conventie is een variabel aantal parameters toegestaan.

DLL-functies aanroepen

U kunt een DLL-functie aanroepen in programmacode of het commandovenster. U roept als volgt een DLL-functie aan:

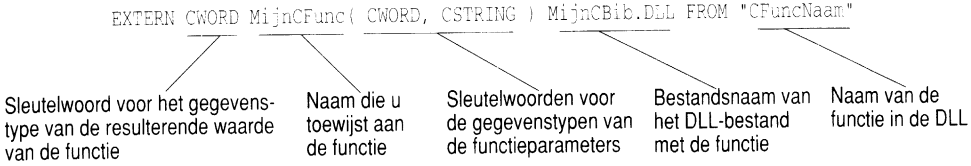
1 Maak een prototype van de niet-dBASE-functie met het commando EXTERN. Een prototype geeft de gegevenstypen aan voor de functieparameters en de resulterende waarde.

2 Roep de niet-dBASE-functie aan in uw dBASE-code.

U kunt hoofdletters en kleine letters gebruiken om niet-dBASE-functies aan te roepen in dBASE, ongeacht de taal waarin de functies zijn geschreven.

Prototypen en conversie van gegevenstypen

In dBASE worden prototypen gebruikt om parameters en resulterende waarden automatisch te converteren tussen dBASE-gegevenstypen en niet-dBASE-gegevenstypen. Gebruik het commando EXTERN om een prototype te definiëren voor elke niet-dBASE-functie, voordat u de functie aanroept. Zie *Commando's en functies* voor meer informatie over het commando EXTERN. In het volgende voorbeeld ziet u een EXTERN-commando.



In Tabel 25.1 ziet u de EXTERN-sleutelwoorden die u gebruikt voor elk gegevenstype.

Tabel 25.1 EXTERN-sleutelwoorden voor parameters en resulterende waarden

	Sleutelwoord	dBASE-gegevenstype	C-gegevenstype	Pascal-gegevenstype	ASM-gegevenstype
Parameters of resultaatwaarden	CDOUBLE	Numeriek	long double (80 bit)	Double	N.v.t.
	CHANDLE	Numeriek	Handles, zoals HANDLE, HWND, HFONT, HDC	Handles, zoals HWND, HFont, HDC	dw
	CINT	Numeriek	int	Integer	dw (16 bit)
	CLOGICAL	Logisch	short int	Integer	dw (16 bit)
	CLONG	Numeriek	long int (32 bit)	Long Int	dd (32 bit)
	CSTRING	Teken	char far * (zero terminated, eindigt op 0)	PChar	dw (16 bit)
	CVOID	N.v.t.	void	Procedure	N.v.t.
	CWORD	Numeriek	short int (16 bit)	WORD	dw (16 bit)
Alleen parameters	CPTR	N.v.t.	void *	Pointer	dd (32 bit)

Dat het nodig is prototypen te gebruiken, wordt duidelijk wanneer u zich het probleem voorstelt om parameters over te brengen naar functies in verschillende talen. De verwachte parameter voor een C-functie is bijvoorbeeld een long int (lang geheel getal) van 32 bits zonder teken. Wanneer u deze functie aanroept in uw dBASE-code, kunt u een numerieke dBASE-variabele overbrengen. Deze parameter wordt automatisch geconverteerd van het dBASE-gegevenstype naar het C-gegevenstype. Resulterende waarden worden eveneens automatisch geconverteerd van C naar dBASE.

Voorbeelden van aanroepen voor DLL-functies

Net als interne dBASE-functies, kunt u niet-dBASE-functies apart of als onderdeel van een dBASE-uitdrukking aanroepen. In het volgende voorbeeld worden beide methoden gedemonstreerd.

```
EXTERN CWORD ACFunc(CWORD, CSTRING) MIJNCBIB.DLL   && Prototype maken van functie
nParam = 10
cParam = "tekst"
nResult = ACFunct(nParam,cParam)                   && ACFunct() uitvoeren
ACFunct(nParam,cParam)                             && Uitvoeren zonder resulterende
                                                   && waarde
```

U kunt ook een interne dBASE-functie of een niet-dBASE-functie aanroepen in een niet-dBASE-functie. In het volgende voorbeeld worden de dBASE-functies CDOW(), DTOC() en DATE() aangeroepen in de Windows-API-functie MessageBox.

```
EXTERN CWORD MessageBox(CHANDLE,CSTRING,CSTRING,CWORD) user.exe
* parameter 1 - CHANDLE - handle van hoofdelementvenster
* parameter 2 - CSTRING - bericht in dialoogvenster
* parameter 3 - CSTRING - titel van dialoogvenster
* parameter 4 - CWORD - stijl van dialoogvenster
MessageBox(0,"Vandaag is het: "+CDOW(DATE()),DTOC(DATE()),65)
```

In het volgende voorbeeld wordt de Windows-API-functie GetFocus aangeroepen in de Windows-API-functie MessageBox.

```
EXTERN CWORD Messagebox(CHANDLE,CSTRING,CSTRING,CWORD) USER.EXE
EXTERN CHANDLE GetFocus(CVOID)  user.exe  && Handle gaat terug naar venster met focus
? Messagebox(GetFocus(),"Hallo ! ", "MessageBox() aanroepen in dBASE",17)
```

U kunt uw eigen namen definiëren voor niet-dBASE-functies met de EXTERN-opdracht. Dit is handig als de niet-dBASE-functie dezelfde naam heeft als een interne dBASE-functie of een functie die u zelf hebt gedefinieerd. U definieert uw eigen naam door de naam op te geven in de EXTERN-opdracht en de werkelijke naam van de functie in het DLL-bestand op te geven met de optie FROM. In het volgende voorbeeld wordt een eigen meldingsvenster gedefinieerd op basis van de Windows-API-functie MessageBox.

```
EXTERN CWORD MijnVenster(CHANDLE,CSTRING,CSTRING,CWORD) USER.EXE FROM "MessageBox"
? MijnVenster(0,"Te uwer informatie: dit is MijnVenster!","Mijn venster",65)
```

Sommige C-functies accepteren venster-handles als parameter of resulteren in venster-handles. Handles zijn nummers die worden toegewezen aan elk venster door de Windows-besturingsomgeving. Sla venster-handles op in dBASE als numerieke variabelen en gebruik CHANDLE om venster-handles op te geven in de EXTERN-opdracht.

Sommige C-functies accepteren een verwijzing naar een reeks (CSTRING) die door de functie wordt gevuld met de resulterende waarde. In dit geval moet u een tekenvariabele overbrengen die groot genoeg is voor het resultaat. Voor bepaalde functies moet u bovendien de maximumlengte van de reeks overbrengen als argument. In het volgende voorbeeld wordt de Windows-API-functie GetSystemDirectory aangeroepen om de naam van de Windows-systeemdirectory op te halen.

```
EXTERN CWORD GetSystemDirectory(CSTRING,CWORD) krnl386.exe   && Prototype
wSysDir = SPACE(144)                                        && Parameter initialiseren
GetSystemDirectory(wSysDir,144)                             && Parameter overbrengen
                                                            && naar functie
? "De Windows-systeemdirectory is: "+wSysDir
```

In het volgende voorbeeld ziet u de dBASE-code voor de aanroep van een C-functie:

```
*******************************************************************************
* dBASE-programma (CVOORBD2.PRG)
* StrRevC():
*     In dit voorbeeld wordt de variabele MijnReeks overgebracht door verwijzing,
*     wordt de reeks vervolgens gewijzigd door de functie StrRevC in de DLL en
*     wordt de gewijzigde reeks daarna weergegeven in dBASE.
*******************************************************************************
EXTERN CVOID StrRevC(CSTRING) CVOORBD2.dll  && Prototype
MijnReeks= "AbCdE"
? "Oorspronkelijke reeks: ",MijnReeks
StrRevC(MijnReeks)
? "Gewijzigd door StrRevC: ",MijnReeks
```

In het volgende voorbeeld ziet u de C-broncode voor de DLL met StrRevC():

```
/*****************************************************************
*  DLL-broncode (CVOORBD2.C)
*****************************************************************/
#include <windows.h>
#include <string.h>
#pragma argsused
int FAR PASCAL LibMain(HINSTANCE hInstance, WORD wDataSeg,
                WORD wHeapSize, LPSTR lpszCmdline){
        if (wHeapSize > 0)
            UnlockData (0) ;
        return 1;}
#pragma argsused
int FAR PASCAL WEP(int wParameter) {
        return 1;}
/*  Function StrRevC()   */
#pragma argsused
void FAR PASCAL StrRevC(LPSTR lpstrString) {
         strrev(lpstrString);}
```

In het volgende voorbeeld ziet u het .DEF-bestand voor de DLL:

```
;*************************************************************
;*  DEF-bestand voor C-DLL (CVOORBD2.DEF)
;*************************************************************
LIBRARY        CVOORBD2
DESCRIPTION    'Een voorbeeld-DLL voor dBASE voor Windows'
EXETYPE        WINDOWS
CODE           PRELOAD MOVEABLE DISCARDABLE
DATA           PRELOAD SINGLE
HEAPSIZE       1400
EXPORTS
               WEP             @1
               STRREVC         @2
```

DLL'S laden en vrijgeven

U moet een DLL-bestand laden in het geheugen voordat u een van de functies in het bestand kunt aanroepen. Wanneer een DLL is geladen, kunt u in dBASE elke functie uit de DLL aanroepen waarvoor u eerder een prototype hebt gemaakt. Een geladen DLL gebruikt slechts een kleine hoeveelheid geheugen, ongeacht de DLL-bestandsgrootte.

Het geheugen wordt automatisch beheerd door de Windows-omgeving. Het aantal DLL-bestanden in het geheugen wordt bijgehouden met een *verwijzingteller*, die telkens wordt verhoogd wanneer een bepaalde DLL wordt geladen door een nieuwe applicatie. Als diverse programma's in dBASE functies aanroepen uit een bepaalde DLL, wordt de verwijzingteller slechts eenmaal verhoogd en wordt het bestand alleen geladen als dit nog niet eerder is geladen.

In dBASE kunnen DLL's worden geladen op drie manieren:

- Een DLL-bestand wordt automatisch geladen wanneer voor het eerst een prototype wordt gemaakt van een functie in de DLL met het commando EXTERN. Dit is de meest gangbare manier om DLL's te laden.

```
EXTERN CWORD MijnCFunc2() MijnCBib.DLL    && Prototype maken van functie en initialiseren
nResult = MijnCFunc2()                    && Functie aanroepen
```

- Laad een DLL-bestand met het commando LOAD DLL, voordat u een van de functies in het bestand aanroept. U kunt LOAD DLL ook gebruiken om bijvoorbeeld te testen op fouten bij het laden van DLL's.

```
LOAD DLL MijnAfb.DLL                        && Initialiseren
DEFINE IMAGE MijnAfb OF MijnForm PROPERTY;  && Afbeeldingsobject voor eerder gemaakt
   DataSource "RESOURCE #1001 MijnAfb.DLL"  && formulier, met DLL-naam en resource-ID
```

- Geef de DLL-naam op in het configuratiebestand van dBASE (DBASEWIN.INI). DLL's die zijn opgegeven in DBASEWIN.INI, worden geladen wanneer u dBASE start. (Zie Appendix C in het *Handboek* voor meer informatie over de instellingen in DBASEWIN.INI.)

```
[LoadDLL]                                 && Begin van DLL-sectie in DBASEWIN.INI
DLL0 = MijnCBib.DLL                       && DLL-bestand dat moet worden geladen
```

U hoeft DLL's niet expliciet vrij te geven. DLL's worden automatisch vrijgegeven wanneer u dBASE afsluit.

Windows-API-functies aanroepen

U roept een Windows-API-functie aan op dezelfde manier als een andere DLL-functie. Er zijn echter twee verschillen:

- De DLL's van de Windows-API worden automatisch geïnitialiseerd wanneer u Windows start. U hoeft dus nooit een Windows-API-DLL te laden met het commando LOAD DLL. U geeft wel de DLL-naam met de API-functie op wanneer u een prototype van de functie maakt met het commando EXTERN.
- De meeste Windows-API-functies maken gebruik van de Pascal-aanroepconventie. Gebruik dus niet de optie CDECL van het commando EXTERN.

In de Windows Software Development Kit (SDK) en diverse boeken van derden worden de Windows-API-functies uitgebreid beschreven en wordt een overzicht gegeven van de functies in elke Windows-DLL. dBASE bevat een header-bestand (WINAPI.H) met EXTERN-prototypen en constantedefinities voor de Windows-API-functies die u het meeste gebruikt. Dit bestand is een hulpmiddel om Windows-API-functies aan te roepen.

U kunt de benodigde prototypen en constanten uit WINAPI.H knippen en in uw programma plakken. Als u veel Windows-API-functies aanroept, kunt u WINAPI.H invoegen in uw programma met de preprocessor-instructie #include. (Zie Hoofdstuk 7 voor meer informatie over preprocessor-instructies.)

In het volgende voorbeeld worden Windows-API-functies gebruikt.

```
EXTERN CLONG GetFreeSpace( CINT )  krnl386.exe        && Resulteert in het aantal
FreeMem=LTRIM(STR(GetFreeSpace(0)/1024))+"K"          && beschikbare bytes systeemgeheugen
? "Vrij systeemgeheugen: "+FreeMem

EXTERN CWORD GetFreeSystemResources( CINT ) user.exe  && Resulteert in het percentage
FreeRes=LTRIM(STR(GetFreeSystemResources(1)))+'%'     && beschikbare systeem-resources
? "Vrije resources: "+FreeRes

EXTERN CWORD GetNumTasks() krnl386.exe                && Resulteert in het aantal
NumTasks = LTRIM(STR(GetNumTasks()))                  && actieve taken in Windows
? "Actieve taken: "+NumTasks
```

De Windows-API-functie MessageBox() is opgeslagen in het bestand USER.EXE. In het volgende voorbeeld wordt het dialoogvenster "Van Windows-API" weergegeven, met het bericht "Hallo allemaal", het stopbordpictogram en twee knoppen, OK en Annuleren. De resulterende waarde is 1 als de knop OK wordt gekozen en 2 als de knop Annuleren wordt gekozen.

```
EXTERN CWORD MessageBox(CHANDLE,CSTRING,CSTRING,CWORD) user.exe
nResponse = MessageBox(0, "Hallo allemaal!", "Van Windows-API", 17)
```

Windows-API-constanten

De Windows-API maakt gebruik van een groot aantal constanten voor eigen stuurelementen, zoals pictogrammen en cursors. Sommige constanten zijn gedefinieerd als gehele getallen, andere als hexadecimale getallen. Met de constante IDC_ARROW

(deze constante is gedefinieerd als 32512) kunt u bijvoorbeeld de pijlvormige cursor opgeven in een functieparameter.

In het volgende voorbeeld worden de constanten SM_CXSCREEN en SM_CYSCREEN gebruikt om de breedte en hoogte van het scherm in te stellen:

```
#define SM_CXSCREEN 0
#define SM_CYSCREEN 1
EXTERN CINT GetSystemMetrics(CINT) user.exe
? "Schermbreedte = ", GetSystemMetrics(SM_CXSCREEN)
? "Schermhoogte = ", GetSystemMetrics(SM_CYSCREEN)
```

Wanneer u constanten gebruikt, moet u soms gehele getallen converteren naar hexadecimale waarden en omgekeerd. dBASE kent hiervoor de volgende conversiefuncties:

- HTOI(). Hiermee converteert u een als tekenreeks opgeslagen hexadecimaal getal naar een geheel getal.
- ITOH(). Hiermee converteert u een geheel getal naar een als tekenreeks opgeslagen hexadecimaal getal.

In het volgende voorbeeld wordt HTOI() gebruikt voor de constante MB_ICONQUESTION om een vraagtekenpictogram weer te geven in een meldingsvenster:

```
#define MB_ICONQUESTION  HTOI("0020")
EXTERN CWORD MessageBox(CHANDLE,CSTRING,CSTRING,CWORD) user.exe
MessageBox(0,"What's up, Doc","Van Windows",MB_ICONQUESTION)
```

Bitbewerking van parameters en resulterende waarden

Bij sommige Windows-API-functies worden afzonderlijke bits geïnterpreteerd in argumenten met gehele getallen of resulterende waarden. Ervaren programmeurs kunnen met de volgende functies bitbewerkingen toepassen op gehele getallen:

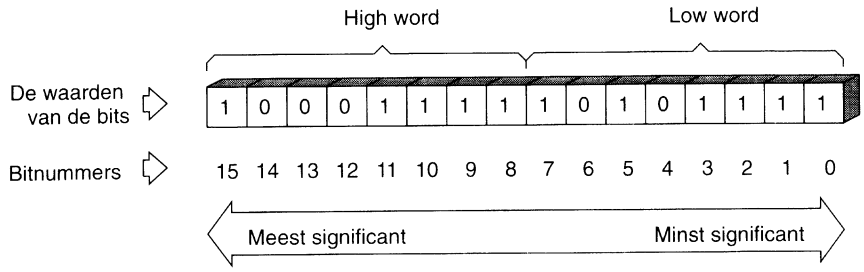
- BITAND(). Hiermee vergelijkt u twee bitwaarden met de logische operator AND.
- BITOR(). Hiermee vergelijkt u twee bitwaarden met de logische operator OR.
- BITXOR(). Hiermee vergelijkt u twee bitwaarden met de logische operator EXCLUSIVE OR
- BITLSHIFT(). Hiermee verschuift u de bits in een geheel getal *n* posities naar links.
- BITRSHIFT(). Hiermee verschuift u de bits in een geheel getal *n* posities naar rechts.
- BITSET(). Hiermee stelt u een bit in (.T. voor aan, .F. voor uit).

Bits in een geheel getal worden genummerd van de minst significante bit (aan de rechterkant) tot de meest significante bit (aan de linkerkant). De bitnummering begint bij 0 voor de minst significante bit. In een geheel getal van 16 bits vormen bits 0 tot en met 7 het low word en bits 8 tot en met 15 het high word.

Windows-API-functies resulteren soms in twee waarden in één geheel getal: één waarde in het low word en de andere waarde in het high word. Met de bitfuncties kunt

u deze waarden ophalen en bewerken. In Afbeelding 25.2 ziet u de bitnummering voor het getal 36783, een 16-bits geheel getal zonder teken.

Afbeelding 25.2 Bitnummering



In het volgende voorbeeld worden bitfuncties en de preprocessor-instructie #define gebruikt om de DOS- en Windows-versies op te halen uit de resulterende waarde van de Windows-API-functie GetVersion().

```
FUNCTION Versies
#define HiWord(x) (BITRSHIFT(BITAND(x,4294901760),16))
#define LoWord(x) (BITAND(x,65535))
#define HiByte(x) (LTRIM(STR(BITRSHIFT(BITAND(x,65280),8))))
#define LoByte(x) (LTRIM(STR(BITAND(x,255))))
EXTERN CLONG GetVersion( ) krnl386.exe
z=GetVersion( )
RETURN "DOS-versie: "+HiByte(HiWord(z))+"."+LoByte(HiWord(z))+ ;
     ", Windows-versie: "+LoByte(LoWord(z))+"."+HiByte(LoWord(z))
```

Zie Hoofdstuk 7 voor meer informatie over de preprocessor-instructie #define.

Hoofdstuk

26

Gegevens uitwisselen met behulp van DDE

Dynamic Data Exchange (DDE) is een Windows-functie waarmee twee toepassingen gegevens kunnen delen en uitwisselen. Hierbij wordt een koppeling tot stand gebracht die dient als communicatiekanaal tussen de toepassingen. De gekoppelde toepassingen hoeven niet dezelfde functie of hetzelfde doel te hebben. Met DDE kunt u bijvoorbeeld gegevens uitwisselen tussen dBASE (een databasebeheersysteem) en Quattro Pro voor Windows (een spreadsheet-toepassing). Het enige vereiste is dat beide toepassingen DDE ondersteunen.

Met een andere functie voor gegevensuitwisseling, Object Linking & Embedding (OLE), kunt u externe toepassingen rechtstreeks gebruiken vanuit een dBASE-tabel of -formulier. Zie het *Handboek* voor meer informatie over OLE.

Werken met DDE

Met DDE kunt u instructies en informatie uitwisselen tussen dBASE en een andere Windows-toepassing via een kanaal dat een *DDE-koppeling* wordt genoemd. Deze tweewegcommunicatie werkt op ongeveer dezelfde manier als een telegraafverbinding, met dBASE aan het ene uiteinde en een andere Windows-toepassing aan het andere uiteinde.

Elke toepassing in de verbinding heeft een aparte functie, afhankelijk van de richting waarin de instructies stromen door de koppeling. Een van beide toepassingen in een DDE-verbinding is de *cliënt* en de andere toepassing is de *server*. De cliënt vraagt gegevens op bij de server, verzendt gegevens naar de server of verzendt opdrachten naar de server die moeten worden uitgevoerd. De server heeft een passievere functie: de server wordt aangeroepen door de cliënt en ontvangt gegevens, produceert gegevens of voert taken uit volgens de instructies van de cliënt. dBASE kan een cliënt, een server of beide tegelijk zijn. dBASE kan dus een andere toepassing aanroepen en instructies geven, of worden aangeroepen door en instructies krijgen van een andere toepassing.

dBASE heeft twee DDE-objectklassen: DDELink en DDETopic. U maakt een DDELink-object wanneer u dBASE wilt gebruiken als cliënt en een DDETopic-object wanneer u dBASE wilt gebruiken als server.

Cliënt-toepassingen maken

Met een DDELink-object start u een sessie in de server-toepassing en brengt u de DDE-koppeling tot stand tussen dBASE en de server. Wanneer de koppeling tot stand is gebracht, kunt u de methoden en kenmerken van het DDELink-object gebruiken om te communiceren met de server-toepassing.

U maakt een DDELink-object met de operator NEW (zie volgende voorbeeld):

```
PUBLIC MijnKoppObj           && MijnKoppObj beschikbaar maken voor andere routines
MijnKoppObj = NEW DDELINK()  && Nieuw DDELink-object maken
```

U kunt een DDELink-object ook maken met het commando DEFINE:

```
DEFINE DDELINK MijnKoppObj   && Nieuw DDELink-object maken
```

In Tabel 26.1 ziet u een overzicht van de kenmerken van de DDELink-klasse.

Tabel 26.1 DDELink-methoden en -kenmerken

Kenmerk	Beschrijving
Advise( )	Hiermee vraagt u de server om de cliënt te melden wanneer een element in het server-document wordt gewijzigd (u maakt een *hot link*, ofwel dynamische koppeling)
ClassName	Hiermee geeft u de DDElink-objectklasse aan
Execute( )	Hiermee verzendt u instructies naar de server in de eigen taal van de server
Initiate( )	Hiermee start u een conversatie met een server-toepassing
OnNewValue	Hiermee voert u een subroutine uit wanneer een element met een hot link in de server-toepassing wordt gewijzigd
Peek( )	Hiermee haalt u een gegevenselement uit een server-document op
Poke( )	Hiermee voegt u een gegevenselement toe in een server-document
Reconnect( )	Hiermee herstelt u een DDE-koppeling die is gesloten met de methode Terminate()
Release( )	Hiermee verwijdert u de DDELink-objectdefinitie uit het geheugen
Server	Hiermee geeft u de naam van de server-toepassing aan
TimeOut	Hiermee stelt u in hoe lang dBASE wacht op een transactie voordat er een fout wordt gemeld
Topic	Hiermee geeft u de naam aan van het object dat u hebt opgegeven met de methode Initiate()
Unadvise( )	Hiermee vraagt u de server om de cliënt niet langer te melden wanneer een element van het server-document wordt gewijzigd (u sluit een hot link)

DDE-koppelingen maken en sluiten

Met het DDELink-object kunt u dBASE (de cliënt) koppelen aan een andere toepassing (de server). De server moet dan wel actief zijn. Gebruik de methode Initiate() van de DDELink-objectklasse om de sessie zo nodig te openen en de koppeling te maken.

```
PUBLIC MijnKoppObj                                    && Beschikbaar maken voor
                                                      && andere routines
MijnKoppObj = NEW DDELink()                           && Nieuw DDELink-object maken

IF MijnKoppObj.Initiate("QPW","C:\QPW\VOORBLD\LENEN.WB1")  && DDE-server-sessie starten
   ? "Verbinding met Quattro geinitialiseerd."
ELSE
   ? "Verbinding met Quattro mislukt."
ENDIF
```

Met deze programmacode probeert u een server-sessie te starten in Quattro Pro voor Windows (als er nog geen sessie actief is), het spreadsheet-bestand LENEN.WB1 te openen en een DDE-koppeling te maken tussen dBASE en Quattro Pro voor Windows. Met het commando IF...ELSE...ENDIF waarmee Initiate() wordt uitgevoerd, geeft u een melding weer in het resultatenpaneel om de gebruiker te laten weten of de poging is gelukt of mislukt.

Wanneer u de DDE-koppeling niet meer nodig hebt, kunt u de koppeling sluiten met de methode Terminate().

```
MijnKoppObj.Terminate()  && DDE-koppeling sluiten
```

Gegevens uitwisselen met server

Wanneer u de koppeling tussen dBASE en de server-toepassing tot stand hebt gebracht, kunt u gegevens opvragen uit het server-document met de methode Peek(), of gegevens verzenden naar het server-document met de methode Poke().

```
PUBLIC MijnKoppObj                                    && Beschikbaar maken voor
                                                      && andere routines
MijnKoppObj = NEW DDELINK()                           && Nieuw DDELink-object maken
MijnKoppObj.Initiate("QPW","C:\QPW\VOORBLD\MIJNWERK.WB1")  && DDE-server-sessie starten
xWaarde = MijnKoppObj.Peek("A:A1")                    && Pagina A, cel A1 bekijken
MijnKoppObj.Poke("A:A1", "12345")                     && "12345" overbrengen naar
                                                      && pagina A, cel A1.
```

Met deze programmacode opent u het bestand MIJNBOEK.WB1. U kopieert de waarde in cel A:A1 naar de geheugenvariabele xWaarde met de methode Peek(). Vervolgens voegt u de tekenreeks "12345" toe in deze cel met de methode Poke(). (Cellokaties in Quattro Pro worden besproken in de volgende sectie.)

Gebieden doorwerken

In het vorige voorbeeld hebt u een tekenreeks verzonden naar cel A:A1 in een spreadsheet-bestand van Quattro Pro. Vaak is de overdracht van één gegevenseenheid echter niet voldoende. Als u bijvoorbeeld de inhoud van meerdere velden en records wilt overbrengen naar een spreadsheet, moet u gegevens schrijven naar diverse cellen. U gebruikt hoervoor een techniek die *region walking*, het doorwerken van gebieden, wordt genoemd.

Bij region walking gaat u van element naar element in een server-document om gegevens te lezen van of schrijven naar elk element. U bestuurt deze verplaatsing door opeenvolgende tekenreeksen te maken die verwijzen naar de elementen. In het

volgende voorbeeld ziet u een programma waarin gebruik wordt gemaakt van region walking.

```
VerzGeg = NEW DDELINK()                                    && Nieuw DDE-object maken
VerzGeg.Initiate("QPW", "C:\QPW\VOORBLD\MIJNBEST.WB1")     && DDE-server-sessie starten
SELECT 1
USE AFNEMERS
RijTel = 1                                          && Beginnen bij celrij 1
DO WHILE RECNO() <= 10                              && Tot en met de eerste 10 records
   xVeld = 1                                        && Beginnen bij eerste veld
   TekTel = ASC("D")                                && Beginnen bij celkolom D
   DO WHILE xVeld <= FLDCOUNT("AFNEMERS")           && Verplaatsen van veld naar veld
      VeldNaam = FIELD(xVeld)                       && Huidige veldnaam ophalen
      CelNaam = "A:"+CHR(TekTel)+LTRIM(STR(RijTel)) && Celidentificatie maken
      VerzGeg.Poke(CelNaam, &VeldNaam)              && Veldinhoud invoegen in cel
      xVeld = xVeld + 1                             && Verplaatsen naar volgende cel
      TekTel = TekTel + 1                           && Verplaatsen naar volgende kolom
   ENDDO
   SKIP                                             && Verplaatsen naar volgende rij
   RijTel = RijTel + 1                              && Eén rij omlaag gaan
ENDDO
```

In deze programmacode gebruikt u de tekenvariabele *CelNaam* als identificatie voor de individuele cellen in een spreadsheet. De variabele bestaat uit drie onderdelen:

- Pagina-identificatie. De letterlijke reeks "D" geeft de eerste pagina in het spreadsheet-bestand aan. (Spreadsheet-bestanden van Quattro Pro bestaan uit diverse niveaus en bevatten pagina's, net als een notitieblok.)
- Kolomletter. Net als bij de meeste andere spreadsheet-toepassingen worden er letters (A, B, C, enzovoort) gebruikt in Quattro Pro om celkolommen aan te geven. In het voorbeeldprogramma wordt de letter van elke volgende kolom gemaakt met de volgende functie-aanroep:

  ```
  CHR(TekTel)
  ```

  TekTel is een numerieke variabele die wordt verhoogd met 1 voor elke volgende kolom. Met de functie CHR() wordt dit nummer geconverteerd naar de equivalente ASCII-letter. *TekTel* wordt geïnitialiseerd met de volgende opdracht:

  ```
  TekTel= ASC("D")
  ```

  Hiermee wordt *TekTel* geïnitialiseerd op 68. Wanneer de waarde 68 wordt overgebracht naar de functie CHR(), resulteert de functie in "D" (de letter van de eerste kolom).
- Rijnummer. In Quattro Pro worden nummers gebruikt om celrijen aan te geven. Elk rijnummer wordt opgeslagen in de variabele *RijTel*. Het rijnummer wordt geconverteerd naar een tekenreeks en ontdaan van voorloopspaties met de volgende functie-aanroep:

  ```
  LTRIM(STR(RijTel))
  ```

Deze drie onderdelen worden samengevoegd in de variabele *CelNaam*. Bij de volgende opdracht gebruikt u *CelNaam* als eerste argument van de methode Poke() om naar de cel te gaan en de waarde in te voegen. De eerste waarde van *CelNaam* is "A:D1", wat

verwijst naar "pagina A, kolom D, rij 1". De volgende waarde van *CelNaam* is "A:E1", wat verwijst naar "pagina A, kolom E, rij 1". Dit proces gaat door totdat alle velden in het eerste record zijn weggeschreven naar de opgegeven cellen.

De sublus wordt beëindigd en de rij (die wordt bepaald door de variabele *RijTel*) wordt verhoogd met 1. *TekTel* wordt opnieuw ingesteld op 68 en het volgende record wordt overgebracht naar de cellen in rij 2. De cel die het eerste veld uit het tweede record ontvangt, is A:D2.

Dit proces gaat door totdat het laatste record naar de spreadsheet is geschreven.

Opdrachten verzenden naar server

DDE biedt meer dan alleen gegevensuitwisseling tussen cliënt en server. U kunt ook de server-toepassing besturen door instructies te verzenden in de taal van de server-applicatie met de methode Execute().

```
PUBLIC MijnKoppObj                                          && Beschikbaar maken voor
                                                            && andere routines
MijnKoppObj = NEW DDELINK()                                 && Nieuw DDE-object maken

MijnKoppObj.Initiate("QPW","C:\QPW\VOORBLD\MIJNBOEK.WB1")   && DDE-server-sessie starten

MijnKoppObj.Execute('{OPEN "C:\DATA.TXT", W}')              && Tekstbestand DATA.TXT maken
MijnKoppObj.Execute("{WRITELN +A:A1}")                      && Cel A:A1 naar DATA.TXT
MijnKoppObj.Execute("{WRITELN +A:A2}")                      && Cel A:A2 naar DATA.TXT
MijnKoppObj.Execute("{WRITELN +A:A3}")                      && Cel A:A3 naar DATA.TXT
MijnKoppObj.Execute("{CLOSE}")                              && Spreadheet-bestand sluiten
```

In deze programmacode start u een sessie in Quattro Pro, opent u spreadsheet-bestand MIJNBOEK.WB1 en maakt u een DDE-koppeling tussen de sessie in Quattro Pro en de huidige dBASE-sessie. Vervolgens gebruikt u de methode Execute() om de volgende bewerkingen uit te voeren:

1. U maakt een nieuw tekstbestand (DATA.TXT) met de Quattro Pro-opdracht {OPEN}.
2. U verzendt de inhoud van de cellen A1, A2 en A3 naar DATA.TXT met de Quattro Pro-opdracht {WRITELN}.
3. U sluit DATA.TXT met de Quattro Pro-opdracht {CLOSE}.

Hot links maken

Een hot link, dynamische koppeling, is een speciale DDE-koppeling waarbij de cliënt-toepassing wordt gewaarschuwd wanneer een opgegeven element in een server-document wordt gewijzigd. U maakt een hot link met de methode Advise() en wijst een codeblok of subroutine toe aan het kenmerk OnNewValue. Het codeblok of de subroutine wordt uitgevoerd wanneer de waarde van het opgegeven element wordt gewijzigd.

```
MijnKoppObj = NEW DDELINK()                                    && Nieuw-DDE object maken
MijnKoppObj.OnNewValue = WaardeAfhandeling                     && dBASE-subroutine
MijnKoppObj.Initiate("QPW","C:\QPW\VOORBLD\MIJNBOEK.WB1")      && DDE-server-sessie starten
MijnKoppObj.Advise("A:A1")                                     && Wijziging melden
* ... rest van het programma wordt uitgevoerd...

FUNCTION WaardeAfhandeling                                     && Routine voor OnNewValue
PARAMETERS Element, Waarde                                     && Automatisch overbrengen
   ? Element," is gewijzigd in ",Waarde                        && Uitvoer genereren
RETURN .T.
```

In deze programmacode gebruikt u de methode Advise () om een hot link te maken naar cel A:A1 van het werkboekbestand MIJNBOEK.WB1 in Quattro Pro. Tijdens het programma wordt u (MijnKoppObj) gewaarschuwd wanneer nieuwe waarden worden ingevoerd in de cel van de server (Quattro Pro). De subroutine of het codeblok dat u opgeeft bij het kenmerk OnNewValue, wordt uitgevoerd wanneer de waarde van cel A:A1 verandert.

Wanneer u de hot link niet meer nodig hebt, sluit u de koppeling met de methode Unadvise().

```
MijnKoppObj.Unadvise("A:A1")
```

Server-toepassingen maken

Net als dBASE een externe toepassing kan gebruiken als DDE-server, kan een externe cliënt-toepassing dBASE gebruiken als DDE-server. Met de cliënt-toepassing kunt u gegevens lezen van en schrijven naar een dBASE-document (meestal een tabel- of querybestand) en instructies verzenden naar de dBASE-sessie.

Wanneer dBASE de server-toepassing is, kunt u een DDETopic-object gebruiken om acties uit te voeren bij gegevensuitwisseling. U kunt dBASE alleen gebruiken als server als u een DDETopic-object hebt gedefinieerd. Het DDETopic-object bepaalt wat er gebeurt wanneer gegevens worden verzonden, gegevens worden opgevraagd of instructies worden gegeven.

U maakt een DDETopic-object met de operator NEW (zie volgende voorbeeld):

```
PUBLIC MijnSrvrObj
MijnSrvrObj = NEW DDETOPIC("MijnObj")  && Variabele is MijnSrvrObj, object is MijnObj
```

U kunt een DDETopic-object ook maken met het commando DEFINE:

```
DEFINE DDETOPIC MijnSrvrObj            && Variabele en object zijn beide MijnSrvrObj
```

Wanneer u de objectverwijzingsvariabele maakt met de operator NEW, geeft u de objectnaam specifiek op (in het vorige voorbeeld is de objectnaam MijnObj). Wanneer u het commando DEFINE gebruikt, krijgt het object dezelfde naam als de objectverwijzingsvariabele.

In Tabel 26.2 ziet u een overzicht van de kenmerken van de DDETopic-klasse.

Tabel 26.2 DDETopic-kenmerken

Kenmerk	Beschrijving
ClassName	Hiermee geeft u de DDETopic-objectklasse aan
Notify( )	Hiermee meldt u de cliënt dat een dBASE-element is gewijzigd
OnAdvise	Hiermee voert u een subroutine uit wanneer een externe toepassing opdracht geeft voor een hot link naar de dBASE-sessie
OnExecute	Hiermee voert u een subroutine uit wanneer een cliënt-toepassing een instructie verzendt naar dBASE
OnPeek	Hiermee voert u een subroutine uit wanneer een cliënt-toepassing opdracht geeft een waarde te lezen van dBASE
OnPoke	Hiermee voert u een subroutine uit wanneer een cliënt-toepassing opdracht geeft een waarde in te voegen in een dBASE-gegevenselement
OnUnadvise	Hiermee voert u een subroutine uit wanneer een cliënt opdracht geeft een hot link te sluiten
Release( )	Hiermee verwijdert u de DDETopic-objectdefinitie uit het geheugen
Topic	Hiermee geeft u het object van het DDELink-object aan.

dBASE starten in cliënt-toepassingen

Een DDETopic-object moet in het geheugen zijn geladen voordat het object kan worden gebruikt. dBASE (de server-toepassing) moet dus een opstartprogramma uitvoeren voordat de DDE-koppeling tot stand wordt gebracht. De volgende handelingen moeten worden uitgevoerd:

1 De cliënt start een dBASE-sessie (als er nog geen sessie actief is).

2 De cliënt geeft dBASE opdracht het opstartprogramma uit te voeren. Dit programma bevat een *intiatie-afhandelingsroutine* (zie volgende sectie).

3 De cliënt initieert de DDE-koppeling met een dBASE-document.

Als u dBASE bijvoorbeeld wilt aanroepen in Quattro Pro voor Windows en een opstartapplicatie wilt uitvoeren in dBASE, kunt u een macroroutine maken in Quattro Pro voor Windows, bijvoorbeeld:

```
{EXEC "DBASEWIN TESTOBJ.PRG", 1}
{INITIATE  "DBASEWIN", "MijnObj", CHANNEL}
```

- TESTOBJ.PRG is de naam van het dBASE-programma met de initiatie-afhandelingsroutine.
- MijnObj is het object dat wordt aangeroepen door Quattro Pro. De cliënt kan een willekeurig object aanroepen, maar roept meestal de naam van een tabel- of querybestand aan.
- CHANNEL is de naam die u toewijst aan een spreadsheet-cel in Quattro Pro. Deze cel bevat een automatisch gegenereerd identificatienummer waarmee de DDE-koppeling wordt onderscheiden van eventuele andere DDE-koppelingen.

De eerste regel van de macrocode bevat de Quattro Pro-opdracht {EXEC}, waarmee u een dBASE-sessie start. De parameter TESTOBJ.PRG geeft aan dat TESTOBJ.PRG moet worden uitgevoerd in dBASE onmiddellijk nadat de dBASE-sessie wordt gestart.

De tweede regel van de macrocode bevat de opdracht {INITIATE}, waarmee een DDE-koppeling wordt gemaakt tussen Quattro Pro voor Windows en de dBASE-sessie.

Initiatie-afhandelingsroutines schrijven

Wanneer u een dBASE-sessie start, maakt u in feite een instantie van een *dBASE-toepassingsobject*. U verwijst naar het toepassingsobject met de systeemgeheugenvariabele _app, die een objectverwijzing bevat.

Wanneer een cliënt een initialisatie-opdracht verzendt (zoals Quattro Pro met de opdracht {INITIATE} in het vorige voorbeeld), wordt de subroutine uitgevoerd die u toewijst aan de actie OnInitiate van het toepassingobject. Met de subroutine voor OnInitiate maakt u een DDETopic-object, waarmee u acties van de DDE-server afhandelt.

In het volgende programma (TESTOBJ.PRG uit het vorige voorbeeld) laat u het _app-object reageren op de actie OnInitiate door een eigen klasse met de naam AANDEELDDEOBJ te maken. U opent ook een tabel (AANDELEN.DBF), waarin het aantal aandelen van diverse bedrijven staat geregistreerd.

```
* TESTOBJ.PRG
PUBLIC Adviseur
_app.DdeServiceName = "AANDEELSERVER"                  && dBASE-sessie een naam geven
_app.OnInitiate = INITAFHANDEL                         && Initiatie-afhandelingsroutine
SET EXACT OFF                                          && Niet exact zoeken
USE AANDELEN ORDER TAG BEDRNAAM                        && Bevat veld AANDELEN
SEEK "MijnBedrijf"                                     && "MijnBedrijf" zoeken

CLASS AANDEELDDEOBJ OF DDETOPIC                        && Eigen klasse maken
   FUNCTION OnPeek (Element)                           && Reageren op leesopdracht
      IF Element = "AANDELEN"
         ? "Aantal aandelen: ", AANDELEN               && Bevestiging
         ? "Aantal aandelen verzenden naar cliënt"     && Bevestiging
         RETURN AANDELEN                               && Aantal verzenden naar cliënt
      ELSE
         ? "Niets verzenden naar cliënt"               && Bevestiging
         RETURN .F.                                    && Onwaar verzenden naar cliënt
      ENDIF

   FUNCTION OnPoke (Element, Waarde)                   && Reageren op invoegopdracht
      ? "Invoegen: ", Element, Waarde                  && Bevestiging
      REPLACE AANDELEN WITH Waarde                     && Wijziging aanbrengen
      IF (Element = Adviseur)                          && Element met hot link
         ? "Wijziging melden aan cliënt "              && Bevestiging
         This.Notify(Adviseur)                         && Melden aan cliënt
      ENDIF
   RETURN .T.
```

```
   FUNCTION OnAdvise (Element)                          && Reageren op hot link
                                                        && maken
      ? "Cliënt melden wanneer", Element, " wijzigt"    && Bevestiging
      Adviseur = Element                                && Naam van element
                                                        && met hot link opslaan
   RETURN .T.

   FUNCTION OnUnadvise (Element)                        && Reageren op sluiten
                                                        && hot link
      IF (Adviseur = Element)                           && Element met hot link
         ? "Niet melden wanneer ",Element," wijzigt"    && Bevestiging
         Adviseur = .F.                                 && Adviseur deactiveren
      ENDIF
   RETURN .T.

   FUNCTION OnExecute (Opd)                             && Reageren op cliënt-opdracht
      ? "Uitvoeren: invoeropdracht is: ",Opd            && Bevestiging
      DO CASE
         CASE Opd= "VERKOPEN"
            REPLACE AANDELEN WITH AANDELEN - 10         && Tien aandelen verkopen
         CASE Opd= "KOPEN"
            REPLACE AANDELEN WITH AANDELEN + 10         && Tien aandelen kopen
         OTHERWISE
            ? "Ontvangen waarde onbekend"               && Waarschuwing
      ENDCASE
      ? "Wijziging van waarde melden aan cliënt"        && Bevestiging
      This.Notify(Adviseur)                             && Wijziging melden aan cliënt
      ? "Uitvoeren: slotwaarde is ", AANDELEN           && Bevestiging
   RETURN .T.
ENDCLASS

FUNCTION InitAfhandel (Object)                          && AANDEELDDEOBJ-object maken
   Object = UPPER(Object)
   xAfhandel = NEW AANDEELDDEOBJ(Object)                && Instantie van AANDEELDDEOBJ
   ? "Server wordt geïnitialiseerd!"                    && Bevestiging
   ? "Mijn object is ", xAfhandel.Object                && Bevestiging
RETURN xAfhandel                                        && Objectverwijzing maken
```

In deze programmacode geeft u subroutines op voor de kenmerken OnPeek, OnPoke, OnAdvise, OnUnadvise en OnExecute van het DDETopic-object. Telkens wanneer u bijvoorbeeld een waarde wegschrijft in Quattro Pro naar een dBASE-element met de opdracht {POKE}, wordt de subroutine voor OnPoke uitgevoerd. In deze subroutine gebruikt u het commando ? om bevestigingsmeldingen weer te geven, en het kenmerk Notify() om door te geven aan Quattro Pro dat het element werd gewijzigd. Bij de subroutine OnInitiate (FUNCTION InitAfhandel) geeft u één parameter op, Object. Deze parameter bevat de naam van het DDE-object dat u hebt opgegeven in de cliënt-toepassing (Quattro Pro voor Windows).

Met het kenmerk DdeServiceName wijst u een naam toe aan de dBASE-sessie. Op deze manier kunt u een bepaalde dBASE-sessie opgeven wanneer meerdere dBASE-sessies actief zijn. In het vorige voorbeeld stelt u DdeServiceName bijvoorbeeld in op

"AANDLSRV", zodat u een DDE-koppeling kunt maken in Quattro Pro naar de juiste dBASE-sessie:

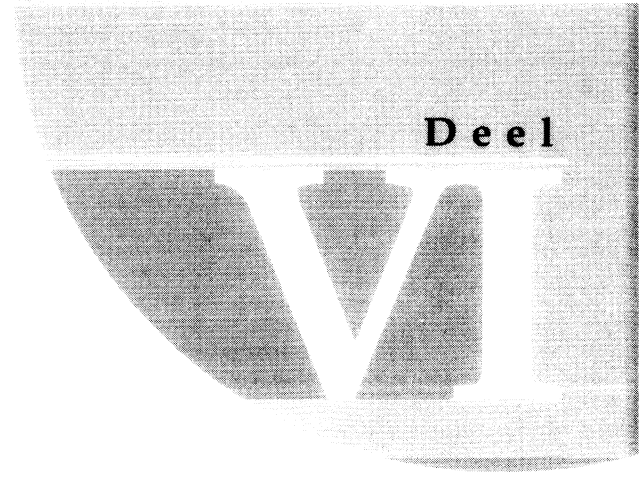
```
_app.DdeServiceName = "AANDLSRV"
```

In de cliënt-toepassing gebruikt u deze naam in plaats van "DBASEWIN" bij een initiatie-opdracht. Als u een DDE-koppeling wilt maken naar de sessie AANDLSRV (in plaats van de andere sessie), geeft u bijvoorbeeld de volgende opdracht {INITIATE} op in Quattro Pro:

```
{INITIATE "AANDLSRV", "MijnObject", CHANNEL}
```

Als u een DDE-koppeling wilt maken naar de andere sessie (waarvan het kenmerk DdeServiceName nog steeds is ingesteld op de standaardwaarde "DBASEWIN"), geeft u bijvoorbeeld de volgende opdracht {INITIATE op in Quattro Pro:

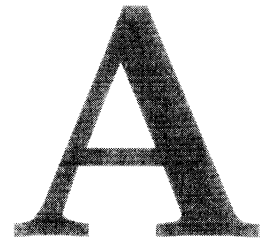
```
{INITIATE "DBASEWIN", "MijnObject", CHANNEL}
```



Appendices

- Appendix A, "dBASE III PLUS- en dBASE IV-applicaties gebruiken"
- Appendix B, "dBASE III PLUS- en dBASE IV-applicaties aanpassen"
- Appendix C, "Werken met tekensets en taalaansturing"

Appendix



dBASE III PLUS- en dBASE IV-applicaties gebruiken

dBASE voor Windows is compatibel met dBASE III PLUS en dBASE IV. De meeste applicaties die zijn geschreven voor dBASE III PLUS en dBASE IV, kunt u gebruiken met een paar kleine wijzigingen in de broncode of zelfs zonder wijzigingen. Start de applicatie en kijk hoe deze werkt.

Opmerking In dit hoofdstuk wordt dBASE III+/IV gebruikt om te verwijzen naar dBASE III PLUS en dBASE IV wanneer deze toepassingen samen worden besproken.

Vanwege de inherente verschillen tussen de tekstgerichte DOS-omgeving en de grafische Windows-omgeving moeten complexe applicaties soms worden aangepast om goed te werken. Bovendien zijn sommige commando's en functies van dBASE III+/IV niet geschikt voor de Windows-omgeving en zijn hiervan nieuwe versies beschikbaar in dBASE voor Windows.

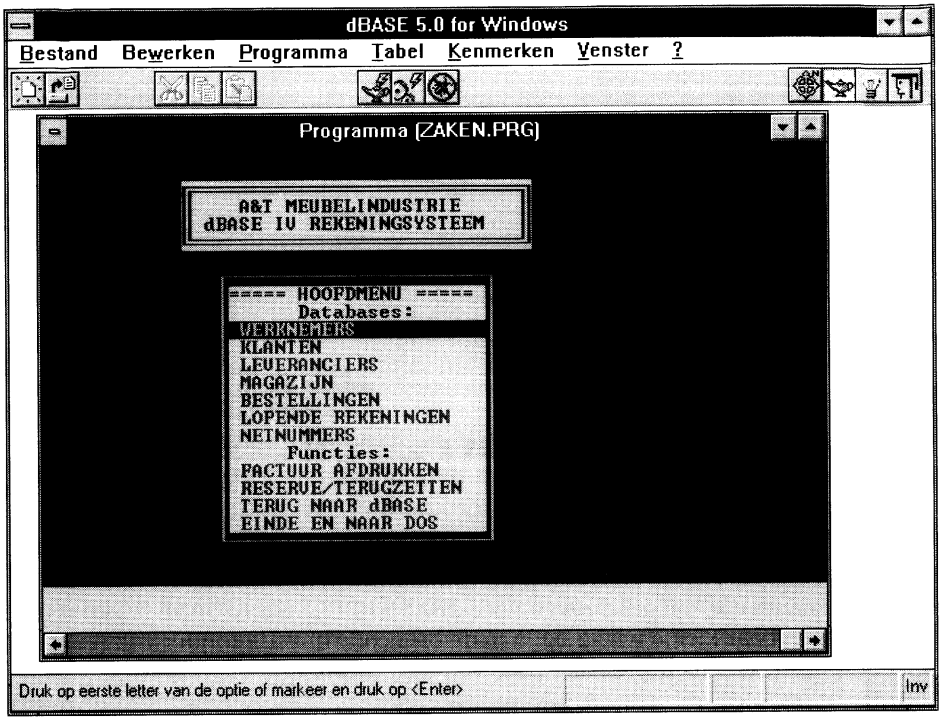
In deze bijlage worden algemene tips gegeven voor het gebruik van dBASE III+/IV-applicaties. Zie Appendix B voor meer informatie over uitbreiding van uw applicaties met functies van dBASE voor Windows.

Bureaublad van dBASE en DOS-applicaties

Wanneer u applicaties start in dBASE III+/IV, nemen de applicaties het volledige scherm in beslag. In dBASE voor Windows worden applicaties weergegeven binnen vensters, en worden dBASE III+/IV-applicaties standaard weergegeven in het resultatenpaneel van het commandovenster.

In Afbeelding A.1 ziet u een voorbeeldapplicatie van dBASE IV (ZAKEN.PRG) tijdens uitvoering in dBASE voor Windows.

Afbeelding A.1 Voorbeeld van een dBASE IV-applicatie tijdens uitvoering in dBASE



Let op de volgende punten:

- In het commandovenster wordt uitvoer standaard weergegeven met een niet-proportioneel font (een font met vaste tekenbreedte). Schermobjecten worden uitgelijnd op dezelfde manier als bij uitvoering in dBASE III+/IV.
- De naam van uw hoofdprogramma wordt weergegeven als titel van het resultatenpaneel in het commandovenster.
- De muis wordt automatisch ondersteund.
- Als u het formaat van het resultatenpaneel in het commandovenster wilt aanpassen aan het aantal regels en kolommen onder DOS, gebruikt u het commando SET DISPLAY in uw programma of wijzigt u het formaat van het resultatenpaneel in het configuratiebestand DBASEWIN.INI (zie Appendix C in het *Handboek*).

Basisstappen

Voer de volgende stappen uit om uw dBASE III+/IV-applicaties te gebruiken in dBASE. In de secties na deze stappen vindt u een nadere uitleg van elke stap.

1 *Ondersteunde bestandstypen.* Kopieer de benodigde DOS-bestanden naar een directory in uw zoekpad, of voeg de directory met de bestanden toe aan het huidige zoekpad voor dBASE.

2 *Windows-omgeving.* Vervang zo nodig de standaardwaarden die zijn ingesteld voor de DOS-omgeving door de corresponderende Windows-instellingen. Zorg dat uw applicaties geen toetsaanslagen vereisen die een vaste betekenis hebben in de Windows-omgeving.

3 *Taaluitbreiding.* Sommige ondersteunde commando's werken anders in dBASE voor Windows dan in dBASE III+/IV. De commando's kunnen andere standaardwaarden, andere toegestane maximum- of minimumwaarden of nieuwe opties hebben. Bestudeer deze sectie om te zien of er wijzigingen zijn die betrekking hebben op uw applicatie.

4 *Verschillen in foutcodes.* De meeste foutcodes in dBASE voor Windows zijn hetzelfde als in dBASE IV, hoewel een aantal foutcodes zijn gewijzigd. Als u het commando ON ERROR gebruikt in uw applicatie, bestudeert u deze sectie om te bepalen of u foutcodes in uw applicatie moet aanpassen.

Ondersteunde bestandstypen

dBASE voor Windows leest broncodebestanden, tabellen en bijbehorende tabelbestanden (zoals indexen) van dBASE III PLUS en dBASE IV. Gecompileerde broncodebestanden (zoals .DBO) en andere bestanden die zijn gegenereerd door dBASE IV-ontwerpfuncties (zoals .SCR) worden niet ondersteund.

In Tabel A.1 ziet u een overzicht van de bestandstypen van dBASE III+/IV die worden ondersteund in dBASE voor Windows. In Appendix A van het *Handboek* worden alle ondersteunde bestandstypen van dBASE III+/IV beschreven. Bovendien kunt u een lijst vinden in Help van alle bestandstypen die worden gebruikt door dBASE voor Windows.

Tabel A.1 Overzicht van ondersteunde dBASE III+/IV-bestandstypen

Bestandstype	Bestandsextensies
Programma's	.PRG (broncodebestanden), .MEM
Gegevens	.DBF, .DBT, .MDX, .NDX
Formulieren	.FMT
Query's	.QBE, .VUE, (.QRY alleen voor dBASE III PLUS)
Rapporten	.FRG, (.FRM alleen voor dBASE III PLUS)
Etiketten	.LBG, (.LBL alleen voor dBASE III PLUS)
Catalogi	.CAT

Kopieer de benodigde bestanden naar een directory in het zoekpad van dBASE voor Windows, of voeg de huidige lokatie van de bestanden toe aan het zoekpad.

Gecodeerde bestanden

dBASE voor Windows ondersteunt geen bestandscodering. Als uw applicatie gecodeerde bestanden gebruikt, moet u de bestanden decoderen voordat u deze gebruikt in dBASE voor Windows.

Windows-omgeving

De Windows-omgeving heeft standaard-resources, zoals printer- en schermaansturingsprogramma's, die kunnen worden gedeeld door Windows-toepassingen. dBASE voor Windows ondersteunt deze gedeelde aansturingsprogramma's zodat u gemakkelijk apparaat-onafhankelijke applicaties kunt schrijven. Daarnaast zijn bepaalde toetsaanslagen gereserveerd in Windows (bijvoorbeeld *Alt+F4*). Gebruik deze toetsaanslagen niet in uw applicaties. Zie de Windows-documentatie voor meer informatie over gereserveerde toetsaanslagen.

DBASEWIN.INI

Sommige dBASE III+/IV-applicaties gebruiken specifieke instellingen die zijn gedefinieerd in het configuratiebestand CONFIG.DB, maar dBASE voor Windows leest dit bestand niet. In plaats daarvan gebruikt dBASE voor Windows een .INI-bestand (DBASEWIN.INI) voor de configuratie. .INI-bestanden zijn tekstbestanden waarin de configuratie-instellingen van Windows-toepassingen zijn opgeslagen. De belangrijkste configuratiebestanden voor de Windows-omgeving zelf zijn WIN.INI en SYSTEM.INI.

Voor bepaalde CONFIG.DB-instellingen bestaan equivalente instellingen in DBASEWIN.INI. Andere instellingen zijn afkomstig uit Windows en zijn niet nodig in DBASEWIN.INI. Als uw applicatie afhankelijk is van instellingen in CONFIG.DB, past u uw code aan of voegt u equivalente instellingen toe aan DBASEWIN.INI. Zie Appendix C in het *Handboek* voor een overzicht van alle DBASEWIN.INI-instellingen.

Syntaxis voor printeraansturingsprogramma's

Als u met de systeemgeheugenvariabele _pdriver verwijst naar printeraansturingsprogramma's van dBASE IV (.PR2-bestanden) in uw programma, moet u de syntaxis bijwerken met het corresponderende Windows-printeraansturingsprogramma. Zie het onderwerp _pdriver in *Commando's en functies* voor meer informatie over de nieuwe syntaxis. In het volgende voorbeeld ziet u hoe u de syntaxis van _pdriver bijwerkt voor een veel gebruikte laserprinter.

Vervang de eerste regel door de tweede regel:

```
_pdriver = "HPLAS260.PR2"

_pdriver = "HPPCL, HP Laser Jet Series II"
```

De variabele _pdriver is de enige systeemgeheugenvariabele voor afdrukken die u moet wijzigen. Andere variabelen, zoals _lmargin en _alignment, worden automatisch toegewezen aan de corresponderende instelling van het Windows-printeraansturingsprogramma.

Zie Hoofdstuk 24 voor meer informatie over afdrukken in applicaties van dBASE voor Windows.

Standaardwaarden voor gegevensindeling

dBASE III+/IV hebben eigen standaardwaarden voor de indeling van datums, tijden, valuta en getallen. dBASE heeft geen ingebouwde standaardwaarden voor deze instellingen. De instellingen worden ingesteld met de optie Internationaal in het Configuratiescherm van Windows. Het Configuratiescherm is een Windows-hulpmiddel om de Windows-omgeving aan te passen en te configureren. Voor de meeste Windows-toepassingen kunt u instellingen opgeven in het Configuratiescherm en hoeft u geen standaardwaarden te coderen in uw programma's.

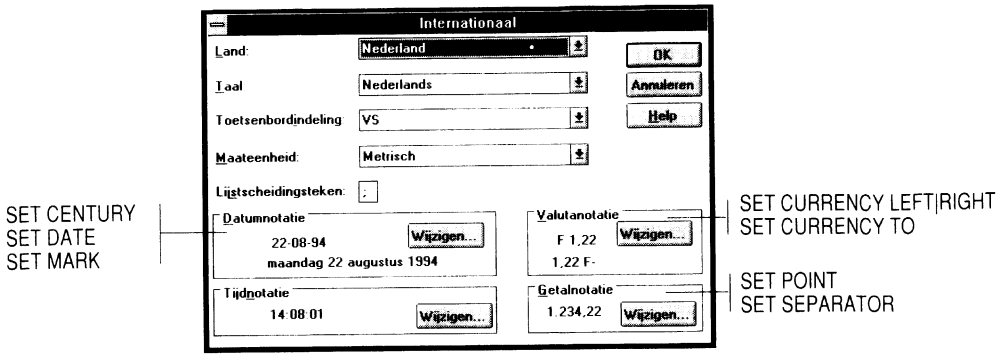
Als u het Configuratiescherm nog nooit hebt gebruikt, kunt u het pictogram voor het Configuratiescherm vinden in de Hoofdgroep. Zie uw Windows-documentatie voor meer informatie over het gebruik van het Configuratiescherm.

Als u een bepaalde standaardinstelling wilt gebruiken in uw applicatie, kunt u de instelling in het Configuratiescherm vervangen op twee manieren:

- U geeft de instellingen op in het configuratiebestand van dBASE voor Windows, DBASEWIN.INI. Zie Appendix C in het *Handboek* voor een overzicht van alle DBASEWIN.INI-instellingen.
- U neemt de gewenste SET-commando's op in uw programma. De volgende SET-commando's vervangen de standaardwaarden in het Configuratiescherm:
  - SET CENTURY. Hiermee deelt u de jaren van datums in.
  - SET CURRENCY LEFT | RIGHT. Hiermee plaatst u het valutateken links of rechts van een geldbedrag.
  - SET CURRENCY TO. Hiermee geeft u het teken of de tekens op die u wilt gebruiken als valutateken.
  - SET DATE. Hiermee deelt u datums in.
  - SET MARK. Hiermee geeft u het teken op dat u wilt gebruiken om dagen, maanden en jaren te scheiden in datums.
  - SET POINT. Hiermee geeft u het teken op dat u wilt gebruiken om decimalen te scheiden van gehele getallen in numerieke waarden.
  - SET SEPARATOR. Hiermee geeft u het teken op dat u wilt gebruiken om elke groep van drie cijfers links van het decimaalteken te scheiden in numerieke waarden die groter zijn dan of gelijk zijn aan 1000.

In Afbeelding A.2 ziet u de relatie tussen deze SET-commando's en de instellingen van de optie Internationaal in het Configuratiescherm van Windows.

Afbeelding A.2 Commando's waarmee de instellingen van de optie Internationaal worden vervangen



Conflicten met Windows-toetsaanslagen

Bepaalde toetscombinaties zijn algemeen geldig voor alle Windows-toepassingen. Met *Alt+F4* sluit u bijvoorbeeld het actieve venster of een dialoogvenster. Wijs geen andere functies toe aan deze gereserveerde toetscombinaties in uw applicatie.

Taaluitbreiding

Sommige dBASE III+/IV-commando's zijn niet meer nodig in de Windows-omgeving en zijn verwijderd. Andere commando's zijn vervangen door nieuwe commando's of functies. Zie Appendix A in *Commando's en functies* voor een overzicht van alle dBASE III+/IV-commando's die niet meer worden ondersteund in dBASE voor Windows.

De commando's die niet worden ondersteund in dBASE voor Windows, veroorzaken geen fouten tijdens compilatie of uitvoering, en hebben geen resultaat in programma's. Deze commando's worden gewoon genegeerd door dBASE voor Windows.

Zelfs als u uw applicaties zonder wijzigingen wilt gebruiken in dBASE voor Windows, kunt u misschien toch de lijst van niet-ondersteunde commando's in *Commando's en functies* eens bekijken. Als een bepaald gedeelte van uw applicatie afhankelijk is van een van deze commando's, kunt u de code aanpassen aan de nieuwe omgeving waarin u uw applicatie gebruikt.

Als u bijvoorbeeld SQL-databases gebruikt in uw applicatie, hebt u misschien de commando's BEGIN TRANSACTION en END TRANSACTION opgegeven om te bepalen wanneer wijzigingen worden weggeschreven naar een record. Omdat het ontwerp van dBASE voor Windows is afgestemd op actiegestuurde programmering, zijn deze commando's vervangen door de functies BEGINTRANS(), COMMIT() en ROLLBACK(). De oude commando's resulteren niet in een fout, maar geven niet hetzelfde resultaat als in dBASE IV.

Functieconflicten

Programmeurs schrijven vaak eigen functies (die "door de gebruiker gedefinieerde functies" of "UDF's" worden genoemd) om taken uit te voeren die niet zijn ingebouwd in de dBASE-taal. Omdat dBASE voor Windows veel nieuwe functies heeft die niet beschikbaar zijn in dBASE III PLUS of dBASE IV, moet u zorgen dat uw UDF's niet dezelfde naam hebben als ingebouwde dBASE-functies. Zie Appendix A in *Commando's en functies* voor een overzicht van de nieuwe functies die zijn toegevoegd aan dBASE.

Specifieke taalwijzigingen

Elk onderwerp in *Commando's en functies* heeft een sectie Overdraagbaarheid, waarin de verschillen tussen dBASE voor Windows en dBASE III+/IV worden beschreven.

Daarnaast kunt u een overzicht vinden in Appendix A van *Commando's en functies* van alle dBASE-commando's die zijn uitgebreid met nieuwe opties ten opzichte van dBASE III+/IV. Bestudeer dit overzicht om de nieuwe mogelijkheden te leren kennen die zijn ingebouwd in commando's die u misschien regelmatig gebruikt. Met deze nieuwe opties kunt u in de toekomst waarschijnlijk nog efficiënter programmeren.

Verschillen in foutcodes

De meeste foutcodes, zoals multi-user-fouten, zijn hetzelfde als in dBASE IV, hoewel sommige foutcodes zijn gewijzigd. Deze wijzigingen zijn onvermijdelijk bij de overgang naar de Windows-omgeving. Als uw programma specifieke runtime-foutcodes opspoort, moet u deze foutcodes misschien bijwerken.

Zie Help voor een overzicht van alle runtime-foutcodes van dBASE.

Appendix

B

dBASE III PLUS- en dBASE IV-applicaties aanpassen

In Appendix A wordt beschreven hoe u dBASE III PLUS- en dBASE IV-applicaties gebruikt in dBASE. Nu u weet hoe u uw programma's kunt starten, wilt u deze waarschijnlijk ook aanpassen, zodat deze eruit zien en werken als Windows-applicaties.

In elk hoofdstuk van dit boek worden programmeerfuncties van dBASE voor Windows beschreven en worden tips gegeven voor dBASE III PLUS- en dBASE IV-programmeurs. In deze bijlage kunt u lezen hoe u een doorsnee dBASE IV-applicatie aanpast, zodat u aan de slag kunt gaan. Zie de corresponderende hoofdstukken elders in dit boek voor meer informatie over de onderwerpen die worden aangesneden in deze bijlage.

Opmerking In deze bijlage wordt dBASE III+/IV gebruikt om te verwijzen naar dBASE III PLUS en dBASE IV wanneer deze toepassingen samen worden besproken.

Zie Appendix A in *Commando's en functies* voor een overzicht van de volgende onderwerpen:

- Nieuwe taalelementen
- Aangepaste taalelementen
- Niet-ondersteunde taalelementen

Het Conversieprogramma

Het Conversieprogramma is een hulpmiddel om de code die is gegenereerd door dBASE IV-ontwerphulpmiddelen, om te zetten in de indeling die wordt gebruikt door de ontwerphulpmiddelen van dBASE voor Windows. Hoewel deze bestanden ook worden uitgevoerd in de huidige staat, kunt u het Conversieprogramma gebruiken om

de bestanden uit te breiden met Windows-stuurelementen, zoals knoppen en keuzelijsten.

Met het Conversieprogramma kunt u indelings-, rapport- en etiketbestanden van dBASE IV gemakkelijk aanpassen aan de Windows-omgeving. Nadat u het Conversieprogramma hebt gebruikt, laadt u de gegenereerde code in de ontwerphulpmiddelen van dBASE voor Windows (zoals Formulierontwerp en Menu-ontwerp).

Het Conversieprogramma is een dBASE voor Windows-programma dat is geïnstalleerd in de subdirectory UTILITY. U start het Conversieprogramma door het programmabestand CB.PRO uit te voeren. Zie de bijbehorende Help voor meer informatie over het gebruik van het Conversieprogramma.

Aanpassingsstrategieën

De bouwblokken van dBASE voor Windows-applicaties zijn formulieren. Formulieren zijn vensters met objecten voor interactie met gebruikers. Uw hoofdtaak bij de aanpassing van een applicatie is het maken van formulieren voor de gebruikersinterface.

Zie het *Handboek* voor meer informatie over het maken van formulieren met Formulierontwerp. Zie Hoofdstukken 13 tot en met 16 in dit boek voor meer informatie over de code die wordt gebruikt door Formulierontwerp.

De benodigde stappen zijn afhankelijk van het type gebruikersinterface dat u wilt ontwikkelen. Er zijn drie typen gebruikersinterface (zie Hoofdstuk 13):

- Een *modale* interface. Deze interface lijkt het meeste op een dBASE III+/IV-applicatie. De gebruiker werkt met één formulier tegelijk en elk formulier neemt de besturing van het bureaublad over. Uw applicatie uitbreiden tot een modale interface is de gemakkelijkste methode.
- Een *niet-modale* interface. Deze interface is volledig actiegestuurd. De gebruiker kan meerdere formulieren of programma's tegelijk gebruiken. Uw applicatie uitbreiden tot een niet-modale interface betekent dat u de code enigzins moet herstructureren om de applicatie volledig actiegestuurd te maken.
- Een *object-georiënteerde,* niet-modale interface. Deze interface is eveneens actiegestuurd. Uw applicatie uitbreiden tot een object-georiënteerde interface betekent dat u uw code omzet in een reeks klassedefinities, die geschikt zijn om opnieuw te gebruiken. Deze methode wordt ook gebruikt door Formulierontwerp, waarbij een klassedefinitie wordt gegenereerd voor elk formulier dat u ontwerpt.

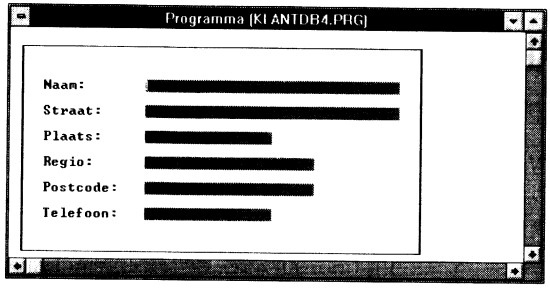
In deze bijlage wordt besproken hoe u uw applicatie uitbreidt tot een modale interface met formulieren. U krijgt eerst een eenvoudige dBASE IV-applicatie te zien, en daarna de aangepaste applicatie voor dBASE voor Windows. Vervolgens worden de belangrijkste verschillen tussen de twee versies beschreven. Zie de vergelijking van de drie typen interfaces in Hoofdstuk 13 voor meer informatie over de uitbreiding van uw applicatie tot een niet-modale of object-georiënteerde interface.

dBASE IV-voorbeeldprogramma

In het voorbeeldprogramma van dBASE IV, KLANTDB4.PRG, wordt een venster weergegeven waarin u een naam kunt opgeven waarnaar u wilt zoeken. Als u niets opgeeft, verdwijnt het venster en stopt het programma. Als u een naam opgeeft, wordt de naam gezocht in de tabel Klanten. Wanneer de naam wordt gevonden, wordt een tweede venster weergegeven waarin u de naam en het adres kunt bewerken.

In Afbeelding B.1 ziet u het programma tijdens uitvoering in het commandovenster van dBASE voor Windows.

Afbeelding B.1 KLANTDB4.PRG tijdens uitvoering in dBASE voor Windows



Dit is de broncode voor KLANTDB4.PRG:

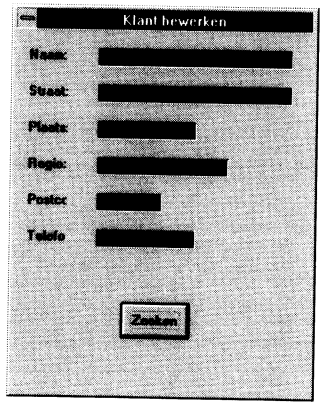
```
* KlantdB4.prg - dBASE IV-editor voor adressenlijst
SET TALK OFF
CLEAR
SET SCOREBOARD OFF
SET STATUS OFF
PUBLIC nNaamLen
nNaamLen= 30

SET VIEW TO KLBEWERK.QBE
DEFINE WINDOW KLBEWERK FROM 1, 1 TO 17, 48
DEFINE WINDOW KLZOEK FROM 5, 1 TO 11, 48
ACTIVATE WINDOW KLZOEK
ZkOpnieuw= .T.
DO WHILE ZkOpnieuw
   cNaam = SPACE(nNaamLen)
   @ 2, 1  SAY "Naam:"
   @ 2, 10 GET cNaam;
            COLOR ,N/GB
   READ
   @ 0, 5 CLEAR TO 0, 40
   ZkOpnieuw = ZOEKNAAM()
ENDDO
DEACTIVATE WINDOW KLZOEK
```

```
*---------------- Procedure/functiedefinities ----------------*
FUNCTION ZOEKNAAM
   IF LEN(TRIM(cNaam)) > 0
      SEEK UPPER(TRIM(cNaam))
      IF .NOT. FOUND()
         ? CHR(7)
         @ 0, 5 SAY "Naam niet gevonden. Probeer het opnieuw."
      ELSE
         DO BEWERKVENST
      ENDIF
      ZkOpnieuw = .T.
   ELSE
      ZkOpnieuw = .F.
   ENDIF
RETURN ZkOpnieuw

PROCEDURE BEWERKVENST
   ACTIVATE WINDOW KLBEWERK
   @ 2, 2  SAY "Naam:"
   @ 2, 14 GET KLANTEN->NAAM;
               COLOR ,N/GB
   @ 4, 2  SAY "Straat:"
   @ 4, 14 GET KLANTEN->STRAAT;
               COLOR ,N/GB
   @ 6, 2  SAY "Plaats:"
   @ 6, 14 GET KLANTEN->PLAATS;
               COLOR ,N/GB
   @ 8, 2  SAY "Regio:"
   @ 8, 14 GET KLANTEN->REGIO;
               COLOR ,N/GB
   @ 10,2  SAY "Postcode:"
   @ 10,14 GET KLANTEN->POSTCODE;
               COLOR ,N/GB
   @ 12,2  SAY "Telefoon:"
   @ 12,14 GET KLANTEN->TELEFOON;
               COLOR ,N/GB
   READ
   DEACTIVATE WINDOW KLBEWERK
RETURN
```

Aangepast dBASE IV-voorbeeldprogramma

KLANTDBW.PRG is een aangepaste versie van KLANTDB4.PRG voor dBASE. In Afbeelding B.2 ziet u het programma tijdens uitvoering op het bureaublad van dBASE voor Windows:

Afbeelding B.2 KLANTDBW.PRG tijdens uitvoering in dBASE voor Windows



Dit is de broncode voor KLANTDBW.PRG:

```
* KlantdBW.prg - dBASE IV-editor voor adressenlijst, aangepast voor dBASE
SET CUAENTER OFF
#define InvoerKleur "N/GB"
#define nNaamLen 30

*----------- Formulier KlZoek maken -----------*
DEFINE FORM KlZoek FROM 11,29.88 TO 18.25,72.20;
   PROPERTY EscExit .T.,;
      Text "Klant zoeken",;
      MDI .F.

DEFINE TEXT TEKST1 OF KlZoek FROM 1,2 TO 1,16;
   PROPERTY ColorNormal "N/W",;
      Text "Naam:"

DEFINE ENTRYFIELD ENTRY1 OF KlZoek FROM 1,18 TO 1,48;
   PROPERTY Value SPACE(nNaamLen),;
      ColorNormal InvoerKleur

DEFINE PUSHBUTTON OKBUTTON OF KlZoek FROM 5,18 TO 6,28;
   PROPERTY  Text "OK",;
      OnClick ZoekNaam

DEFINE PUSHBUTTON CANCELBUTTON OF KlZoek FROM 5,30 TO 6,40;
   PROPERTY Text "Annuleren",;
      OnClick {;CLOSE FORM KlZoek}
*--------- Einde van Formulier KlZoek ---------*
```

```
*------------ Formulier KlBewerk maken ------------*
DEFINE FORM KlBewerk FROM 12.5, 24.5 TO 31.5, 70.38;
   PROPERTY EscExit .T.,;
      View "KLBEWERK.QBE",;
      Text "Klant bewerken",;
      MDI .F.

   DEFINE TEXT TEKST1 OF KlBewerk FROM 1,2 TO 1,15;
      PROPERTY Text "Naam:"

   DEFINE ENTRYFIELD ENTRY1 OF KlBewerk AT 1,18;
      PROPERTY DataLink "KLANTEN->NAAM",;
         ColorNormal InvoerKleur

   DEFINE TEXT TEKST1 OF KlBewerk FROM 3,2 TO 3,15;
      PROPERTY Text "Straat:"

   DEFINE ENTRYFIELD ENTRY2 OF KlBewerk AT 3,18;
      PROPERTY  DataLink "KLANTEN->STRAAT",;
         ColorNormal InvoerKleur

   DEFINE TEXT TEKST3 OF KlBewerk FROM 5,2 TO 5,15;
      PROPERTY Text "Plaats:"

   DEFINE ENTRYFIELD ENTRY3 OF KlBewerk AT 5,18;
      PROPERTY DataLink "KLANTEN->PLAATS",;
         ColorNormal InvoerKleur

   DEFINE TEXT TEKST4 OF KlBewerk FROM 7,2 TO 7,15;
      PROPERTY Text "Regio:"

   DEFINE ENTRYFIELD ENTRY4 OF KlBewerk AT 7,18;
      PROPERTY DataLink "KLANTEN->REGIO",;
         ColorNormal InvoerKleur

   DEFINE TEXT TEKST5 OF KlBewerk FROM 9,2 TO 9,15;
      PROPERTY Text "Postcode:"

   DEFINE ENTRYFIELD ENTRY5 OF KlBewerk AT 9,18;
      PROPERTY DataLink "KLANTEN->POSTCODE",;
         ColorNormal InvoerKleur

   DEFINE TEXT TEKST6 OF KlBewerk FROM 11,2 TO 11,15;
      PROPERTY Text "Telefoon:"

   DEFINE ENTRYFIELD ENTRY6 OF KlBewerk AT 11,18;
      PROPERTY DataLink "KLANTEN->TELEFOON",;
         ColorNormal InvoerKleur

   DEFINE PUSHBUTTON ZOEKKNOP OF KlBewerk FROM 15,17 TO 16,27;
      PROPERTY Text "Zoeken",;
         OnClick ZOEKKNOP_OnClick
```

```
*---------- Einde van formulier KlBewerk ----------*

KlBewerk.ReadModal()

*-------------- Proceduredefinities --------------*
   PROCEDURE ZOEKKNOP_OnClick
      KlZoek.READMODAL()
      KlBewerk.Entry1.SetFocus()
   RETURN

   PROCEDURE ZoekNaam
      IF .NOT. EMPTY(form.Entry1.Value)
         SEEK(UPPER(TRIM(form.Entry1.Value)))
         IF .NOT. FOUND()
            ? CHR(7)
            form.StatusMessage = "Naam niet gevonden. Probeer het opnieuw"
            form.Entry1.SetFocus()
         ELSE
            form.StatusMessage = ""
            form.Entry1.Value = SPACE(nNaamLen)
            CLOSE FORM KlZoek
         ENDIF
      ELSE
         form.StatusMessage = ""
         CLOSE FORM KlZoek
      ENDIF
   RETURN
```

Codeverschillen

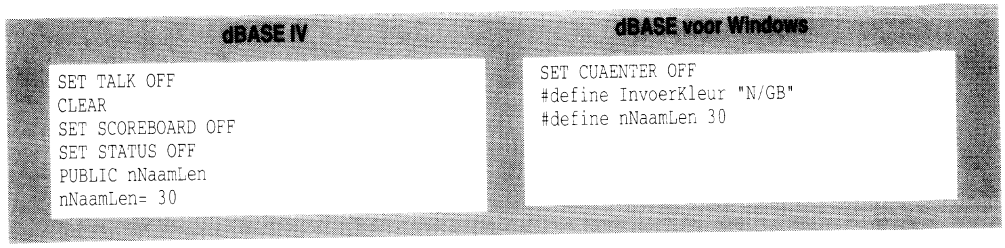
In deze sectie worden bepaalde code-onderdelen van de twee voorbeeldprogramma's in de vorige secties besproken en vergeleken. Hierbij worden de verschillen in code tussen beide programma's aangegeven en wordt vermeld waar u meer informatie kunt vinden elders in dit boek.

Initialisatie

De meeste dBASE IV-programma's beginnen met opdrachten om PUBLIC-variabelen te initialiseren en de omgeving in te stellen. dBASE voor Windows-applicaties gebruiken weinig algemene variabelen. De meeste algemene instellingen kunt u opgeven met objectkenmerken en sommige algemene instellingen worden niet ondersteund.

De dBASE IV-applicatie begint met diverse SET-commando's en de definitie van een PUBLIC-variabele. De dBASE voor Windows-applicatie begint met preprocessor-instructies en één SET-commando.

Afbeelding B.3 Vergelijking van initialisatiecodes



- In dBASE IV wordt commando-uitvoer uitgeschakeld met SET TALK OFF en wordt het scherm gewist met CLEAR. In dBASE voor Windows verschijnt commando-uitvoer alleen in het resultatenpaneel van het commandovenster. Omdat formulieren verschijnen op het bureaublad, zijn deze commando's niet nodig.
- In de dBASE IV-applicatie wordt de weergave van statusinformatie uitgeschakeld met SET SCOREBOARD OFF en SET STATUS OFF. Statusinformatie verschijnt in een adresseerbaar gebied van het scherm, waardoor er conflicten kunnen ontstaan met schermuitvoer. In dBASE voor Windows verschijnt deze informatie op de statusbalk en kunnen er geen conflicten onstaan met schermuitvoer. Deze commando's worden dan ook niet ondersteund (SET STATUS bestaat wel in dBASE, maar heeft een gewijzigde functie).
- In Windows-toepassingen worden formulieren voorgelegd wanneer u drukt op *Enter*. *Enter* werkt hetzelfde als *Ctrl+W* in dBASE III PLUS of *Ctrl+End* in dBASE IV. Met het nieuwe dBASE-commando SET CUAENTER OFF wordt de werking van Enter ingesteld op de werking van *Enter* in dBASE III+/IV, waarbij de focus wordt verplaatst naar het volgende beschikbare stuurelement.

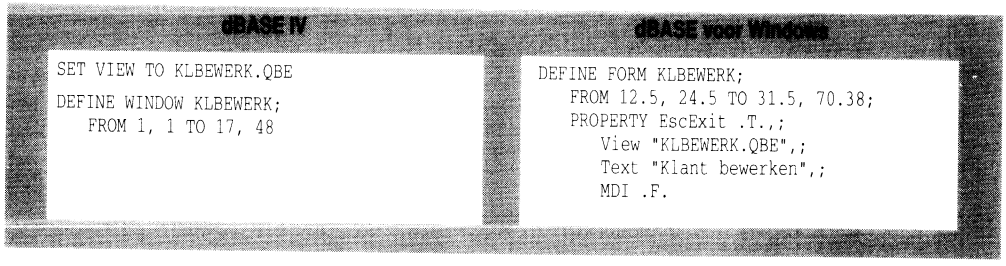
  In het algemeen kunt u het beste de standaardinstelling ON (aan) voor SET CUAENTER handhaven. Als u echter de overgang naar Windows-toetsaanslagen wilt vergemakkelijken, kunt u SET CUAENTER instellen op OFF aan het begin van uw aangepaste programma's. Zie Appendix B in het *Handboek* voor een overzicht van de toetsaanslagen die worden gebruikt in dBASE voor Windows.
- In de dBASE IV-applicatie wordt *nNaamLen* geïnitialiseerd als een PUBLIC-variabele. In dBASE voor Windows worden algemene constanten meestal gedefinieerd met de preprocessor-instructie #define. Zie Hoofdstuk 7 voor meer informatie over preprocessor-instructies.

Vensterdefinitie

In dBASE IV-applicaties worden soms vensters gemaakt om gegevens weer te geven. In dBASE voor Windows worden formulieren gebruikt voor interactie met gebruikers. Formulieren zijn Windows-vensters met gebruikersinterface-objecten. Zie Hoofdstuk 13 voor meer informatie over programmeren met verschillende typen formuliervensters.

In de dBASE IV-applicatie wordt een venster gemaakt om klanten te bewerken. In de dBASE voor Windows-applicatie wordt hiervoor een formulier gemaakt. Beide applicaties gebruiken KLBEWERK.QBE om de tabelomgeving in te stellen.

Afbeelding B.4 Vergelijking van vensterdefinities



- Vensters die worden gemaakt met DEFINE WINDOW, verschijnen in het resultatenpaneel van het commandovenster. Formuliervensters die worden gemaakt met DEFINE FORM, verschijnen op het bureaublad.
- Met het dBASE IV-commando DEFINE WINDOW wordt een venster gemaakt waarin uitvoer van bijvoorbeeld het commando @...SAY verschijnt. Met het dBASE voor Windows-commando DEFINE FORM wordt een formuliervenster gemaakt met veel kenmerken die u kunt instellen. In het dBASE voor Windows-voorbeeldprogramma worden de volgende formulierkenmerken ingesteld:
  - EscExit. Dit kenmerk is ingesteld op .T. (waar). U kunt dan het formulier sluiten door te drukken op *Esc*.
  - Text. Hiermee wordt de titel ingesteld die verschijnt op de titelbalk van het venster.
  - MDI. Dit kenmerk moet worden ingesteld op .F. (onwaar) voor modale vensters. Zie Hoofdstuk 13 voor meer informatie over de eigenschappen van MDI-vensters.
- Met het dBASE IV-commando SET VIEW wordt de tabelomgeving uit een .QBE-bestand ingesteld. In dBASE voor Windows wordt hetzelfde .QBE-bestand gekoppeld aan het formulier met het formulierkenmerk View.

SAY/GET en gegevensobjecten

In dBASE IV-applicaties wordt het commando @...GET gebruikt om velden weer te geven en het commando @...SAY om labels voor de velden te genereren. In dBASE voor Windows-applicaties worden invoervakken en andere objecten gebruikt om velden weer te geven, en worden tekstobjecten gebruikt om labels voor de velden te genereren. Zie de hoofdstukken over Formulierontwerp in het *Handboek* voor meer informatie over de plaatsing van objecten in formulieren.

Beide applicaties geven velden van de tabel Klanten weer. In de dBASE IV-applicatie worden de velden weergegeven in het venster KLBEWERK en in de dBASE voor Windows-applicatie worden de velden weergegeven in het formulier KlBewerk.

Afbeelding B.5 Vergelijking van SAY/GET en gegevensobjecten

dBASE IV

```
ACTIVATE WINDOW KLBEWERK
@ 2, 2  SAY "Naam:"
@ 2, 14 GET KLANTEN->NAME;
      COLOR ,N/GB
@ 4, 2  SAY "Straat:"
@ 4, 14 GET KLANTEN->STRAAT;
      COLOR ,N/GB
@ 6, 2  SAY "Plaats:"
@ 6, 14 GET KLANTEN->PLAATS;
      COLOR ,N/GB
@ 8, 2  SAY "Regio:"
@ 8, 14 GET KLANTEN->REGIO;
      COLOR ,N/GB
@ 10,2  SAY "Postcode:"
@ 10,14 GET KLANTEN->POSTCODE;
      COLOR ,N/GB
@ 12,2  SAY "Telefoon"
@ 12,14 GET KLANTEN->TELEFOON;
      COLOR ,N/GB
```

dBASE voor Windows

```
DEFINE TEXT TEKST1 OF KlBewerk FROM 2,2 TO 2,15;
   PROPERTY Text "Naam:"
DEFINE ENTRYFIELD ENTRY1 OF KlBewerk AT 2,18;
   PROPERTY DataLink "KLANTEN->NAAM",;
   ColorNormal InvoerKleur
DEFINE TEXT TEKST1 OF KlBewerk FROM 4,2 TO 4,15;
   PROPERTY Text "Straat:"
DEFINE ENTRYFIELD ENTRY2 OF KlBewerk AT 4,18;
   PROPERTY DataLink "KLANTEN->STRAAT",;
   ColorNormal InvoerKleur
DEFINE TEXT TEKST3 OF KlBewerk FROM 6,2 TO 6,15;
   PROPERTY Text "Plaats:"
DEFINE ENTRYFIELD ENTRY3 OF KlBewerk AT 6,18;
   PROPERTY DataLink "KLANTEN->PLAATS",;
      ColorNormal InvoerKleur
DEFINE TEXT TEKST4 OF KlBewerk FROM 8,2 TO 8,15;
   PROPERTY Text "Regio:"
DEFINE ENTRYFIELD ENTRY4 OF KlBewerk AT 8,18;
   PROPERTY DataLink ""KLANTEN->REGIO",;
      ColorNormal InvoerKleur
DEFINE TEXT TEKST5 OF KlBewerk FROM 10,2 TO
10,15;
   PROPERTY Text "Postcode:"
DEFINE ENTRYFIELD ENTRY5 OF KlBewerk AT 10,18;
   PROPERTY DataLink "KLANTEN->POSTCODE",;
      ColorNormal InvoerKleur
DEFINE TEXT TEKST6 OF KlBewerk FROM 12,2 TO
12,15;
   PROPERTY Text "Telefoon:"
DEFINE ENTRYFIELD ENTRY6 OF KlBewerk AT 12,18;
   PROPERTY DataLink "KLANTEN->TELEFOON",;
      ColorNormal InvoerKleur
```

- In dBASE IV verschijnen de commando's @...SAY/GET in het actieve venster (indien aanwezig). De dBASE IV-applicatie activeert het venster KLBEWERK. In dBASE voor Windows verschijnen de invoervakken, teksten en alle andere objecten voor de gebruikersinterface in een formulier. Met de optie OF van het commando DEFINE geeft u op in welk formulier de objecten verschijnen. In de dBASE voor Windows-applicatie wordt het formulier KlBewerk gedefinieerd.
- In dBASE IV geeft u de veldnaam op na GET. In dBASE voor Windows koppelt u een invoervak aan een tabelveld met het kenmerk DataLink.
- In dBASE IV geeft u de coördinaten op na @. In dBASE voor Windows geeft u de coördinaten op na de optie AT. U kunt de coördinaten ook opgeven met de kenmerken Top en Left.

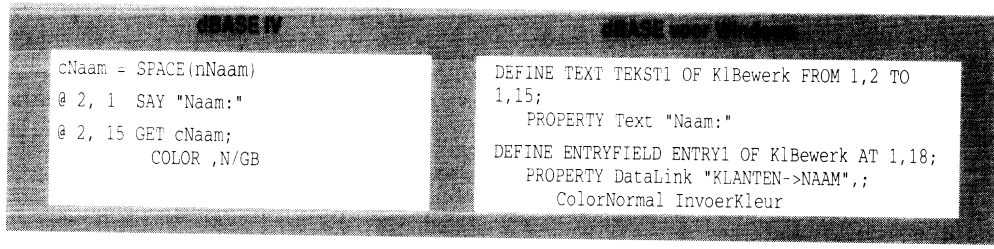
- De opties PICTURE en COLOR in dBASE IV corresponderen met de kenmerken Picture en ColorNormal in dBASE voor Windows.

Geheugenvariabelen

Het gebruik van geheugenvariabelen is in belangrijke mate aangepast in dBASE voor Windows. Voor de invoer en weergave van gegevens gebruikt u het kenmerk Value van objecten in plaats van algemene variabelen. Wanneer u toch traditionele geheugenvariabelen wilt gebruiken, kunt u deze effectiever beheren door twee nieuwe typen variabelen te gebruiken: LOCAL en STATIC. Zie Hoofdstuk 5 voor meer informatie over de verschillende typen variabelen.

In beide applicaties wordt de gebruiker gevraagd een naam op te geven waarnaar wordt gezocht.

Afbeelding B.6 Vergelijking van geheugenvariabelen



In de dBASE IV-applicatie wordt een geheugenvariabele geïnitialiseerd en wordt de variabele weergegeven met het commando @...GET. In de dBASE voor Windows-applicatie wordt geen geheugenvariabele gemaakt, maar wordt het kenmerk Value van een invoervakobject gebruikt. Als u een waarde invoert, wordt deze waarde automatisch opgeslagen in het kenmerk Value.

Als u kenmerk Value gebruikt in plaats van een traditionele variabele, worden de gegevens vastgelegd in het formulier en kunt u conflicten bij de naamgeving van variabelen voorkomen. Zie Hoofdstukken 9 tot en met 12 voor meer informatie over de voordelen van object-georiënteerd coderen.

Procedures en functies

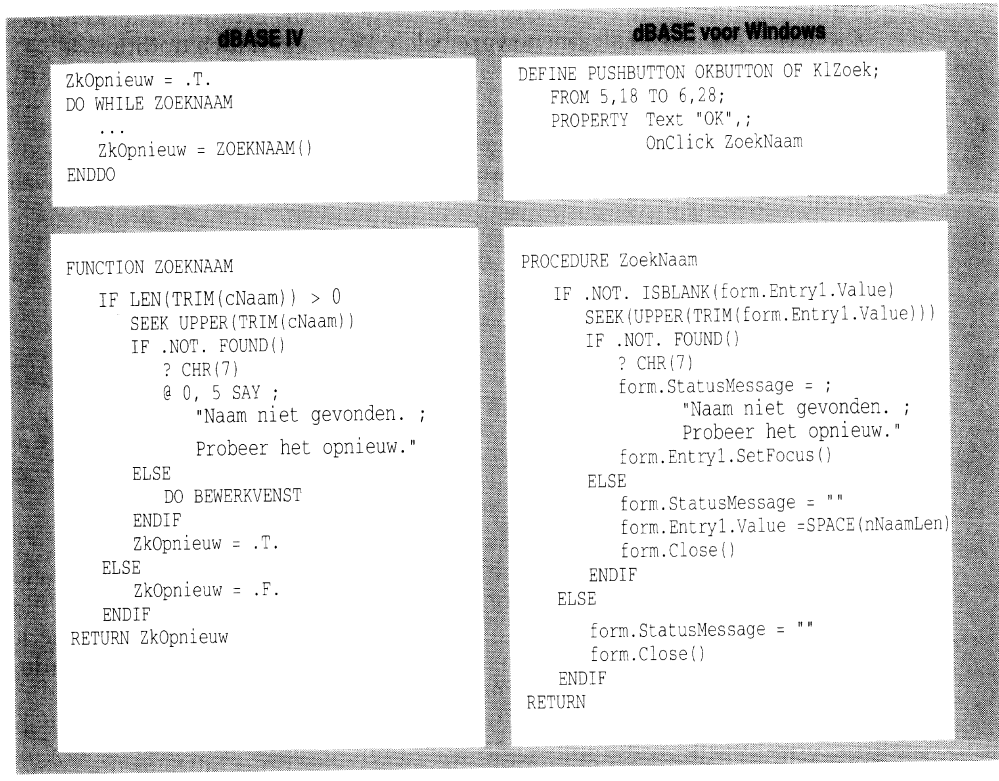
De code voor procedures en functies (de kern van de meeste dBASE-programma's) komt grotendeels overeen in dBASE IV en dBASE voor Windows. De voornaamste verschillen zijn de manier waarop procedures en functies worden aangeroepen en de manier waarop er wordt verwezen naar gegevens.

Omdat dBASE voor Windows-applicaties actiegestuurd zijn, worden procedures en functies meestal aangeroepen wanneer acties plaatsvinden en worden deze niet expliciet aangeroepen door een programma. Bovendien worden veel gegevenselementen opgeslagen als kenmerken van objecten. In subroutines worden "stippen" gebruikt om te verwijzen naar kenmerken.

Zie Hoofdstuk 2 voor een inleiding tot actiegestuurd programmeren, Hoofdstuk 14 voor meer informatie over het schrijven van actie-afhandelingsroutines, en Hoofdstuk 9 voor een inleiding tot het gebruik van objecten en kenmerken.

In beide applicaties wordt een subroutine gebruikt om de naam op te zoeken.

Afbeelding B.7 Vergelijking van aanroep en definitie van subroutines



- In de dBASE IV-applicatie wordt de functie ZOEKNAAM() expliciet aangeroepen in de code. In de dBASE voor Windows-applicatie wordt ZOEKNAAM omgezet in een actie-afhandelingsroutine, die wordt toegewezen aan het kenmerk OnClick van een knop. Wanneer u klikt op de knop, wordt de actie-afhandelingsroutine gestart.
- Beide applicaties gebruiken SEEK om de naam op te zoeken. In de dBASE IV-applicatie wordt de variabele *cNaam* gezocht. In de dBASE voor Windows-applicatie wordt het kenmerk Value van het invoervak Entry1 gezocht. De uitdrukking `form.Entry1.Value` verwijst naar het kenmerk Value van het object Entry1 in het huidige formulier.
- Als de naam niet wordt gevonden, verschijnt in beide applicaties een melding. In de dBASE IV-applicatie wordt het commando @...SAY gebruikt. In de dBASE voor Windows-applicatie wordt het formulierkenmerk StatusMessage ingesteld, waarmee het bericht wordt weergegeven op de statusbalk.

- Als de naam wordt gevonden, verschijnt het bewerkvenster in beide applicaties. In de dBASE IV-applicatie wordt het venster KLBEWERK weergegeven door expliciet de procedure BEWERKVENST aan te roepen. In de dBASE voor Windows-applicatie is het formulier KlBewerk al geopend. Het huidige formulier wordt gesloten met `form.Close()`, waarmee de focus wordt verplaatst naar KlBewerk.

Menu's

dBASE voor Windows biedt een krachtige nieuw hulpmiddel om standaardmenu's in Windows-stijl te maken voor applicaties. De menu-opdrachten van dBASE IV worden ondersteund (echter niet in formulieren). Als u menu's wilt maken, gebruikt u Menu-ontwerp. Dit hulpmiddel is een onderdeel van Formulierontwerp. Menu-ontwerp genereert een .MNU-bestand met de dBASE-code om een menu te maken. U koppelt .MNU-bestanden aan formulieren met het formulierkenmerk MenuFile.

Als u dBASE IV-menucode wilt omzetten in dBASE voor Windows-menucode, gebruikt u het Conversieprogramma. Het Conversieprogramma maakt een .MNU-bestand.

In Tabel B.1 ziet u een overzicht van de menu-opdrachten van dBASE IV en de corresponderende handelingen in dBASE voor Windows. Zie Hoofdstuk 16 voor meer informatie over het maken van menu's.

Tabel B.1 Vergelijking van manieren om menu's te maken

	dBASE IV		
Menutaak	**Horizontale balk**	**Popup**	**dBASE voor Windows**
Definiëren	DEFINE MENU DEFINE PAD	DEFINE POPUP DEFINE BAR	U maakt objecten voor alle typen menu's met de klasse MENU.
Acties toewijzen	ON MENU ON PAD ON SELECTION MENU ON SELECTION PAD ON EXIT MENU ON EXIT PAD	ON POPUP ON BAR ON SELECTION POPUP ON SELECTION BAR ON EXIT POPUP ONEXIT BAR	In dBASE gaat u anders te werk. Als u acties wilt toewijzen, koppelt u actie-afhandelingsroutines aan de actie OnClick van elk menu, of verwerkt u alle menuselecties met een actie-afhandelingsroutine die u koppelt aan het formulierkenmerk OnSelection.
Activeren of weergeven	ACTIVATE MENU SHOW MENU	ACTIVATE POPUP SHOW POPUP	Als u het formulier opent, wordt automatisch het formuliermenu geactiveerd.
Inactief maken	DEACTIVATE MENU RELEASE MENUS CLEAR MENUS	DEACTIVATE POPUP RELEASE POPUPS CLEAR POPUPS	Het volledige menu is actief zolang het formulier open is. U kunt een individuele menu-optie inactief maken met het bijbehorende kenmerk Enabled.

In Afbeelding B.8 ziet u een vergelijking tussen menucode in dBASE IV en vergelijkbare code in dBASE voor Windows. Met de dBASE IV-code, die afkomstig is uit de voorbeeldapplicatie ZAKEN.PRG, maakt u een popup-menu. Met de dBASE voor

Windows-code, die is geconverteerd van ZAKEN.PRG, wordt een hiërarchie van menu-objecten gemaakt.

Afbeelding B.8 Vergelijking van menucodes

dBASE IV

```
DEFINE POPUP Mainmenu FROM 7,27 TO 22,50

DEFINE BAR  1 OF Mainmenu;
   PROMPT "===== HOOFDMENU =====" SKIP

DEFINE BAR  2 OF Mainmenu ;
   PROMPT "      Databases:" SKIP

DEFINE BAR  3 OF Mainmenu;
   PROMPT " WERKNEMERS"

DEFINE BAR  4 OF Mainmenu;
   PROMPT " KLANTEN"

DEFINE BAR  5 OF Mainmenu;
   PROMPT " LEVERANCIERS"

DEFINE BAR  6 OF Mainmenu;
   PROMPT " MAGAZIJN"

DEFINE BAR  7 OF Main;
   PROMPT " BESTELLINGEN"

DEFINE BAR  8 OF Main;
   PROMPT " LOPENDE REKENINGEN"

DEFINE BAR  9 OF Main;
   PROMPT " NETNUMMERS"
```

dBASE voor Windows

```
DEFINE MENU WeergvZaken OF f.Mainmenu;
   PROPERTY Text "&Weergeven"

DEFINE MENU Werknemers OF f.Mainmenu.WeergvZaken;
   PROPERTY  Text "W&erknemers",;
             OnClick WerknemersWeergv

DEFINE MENU Klanten OF f.Mainmenu.WeergvZaken;
   PROPERTY Text "&Klanten",;
             OnClick KlantenWeergv

DEFINE MENU Leveranciers OF f.Mainmenu.WeergvZaken;
   PROPERTY Text "&Leveranciers",;
             OnClick LeverncrWeergv

DEFINE MENU Magazijn OF f.Mainmenu.WeergvZaken;
   PROPERTY Text "&Magazijn",;
             OnClick MagaznWeergv

DEFINE MENU LopndRekning OF f.Mainmenu.WeergvZaken;
   PROPERTY Text "L&opende rekeningen",;
             OnClick LRWeergv

DEFINE MENU Netnummers OF f.Mainmenu.WeergvZaken;
   PROPERTY Text "&Netnummers",;
             OnClick NetnrWeergv
```

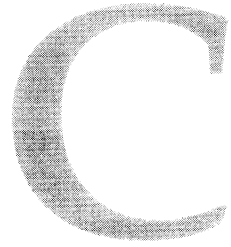
- In de dBASE IV-code wordt de optie OF gebruikt om het popup-menu voor elke menubalk op te geven. In de dBASE voor Windows-code wordt de optie OF gebruikt om de plaats van elk menu-object in de menuhiërarchie aan te geven. Met de uitdrukking `f.Mainmenu.WeergvZaken` wordt het nieuwe object toegevoegd aan het menu WeergvZaken in het menu Mainmenu van het huidige formulier.
- In dBASE IV moet u het commando ON BAR of ON SELECTION BAR gebruiken om een procedure op te geven die moet worden uitgevoerd. In dBASE voor Windows gebruikt u het kenmerk OnClick van elk menu om de procedure op te geven die moet worden uitgevoerd.
- In de dBASE IV-code wordt de tekst van het menu opgegeven met de optie PROMPT. In dBASE voor Windows geeft u de tekst van het menu op bij het kenmerk Text van het menu-object.
- In de dBASE-code wordt een en-teken (&) gebruikt om een keuzeletter op te geven voor het menu. U drukt op *Alt+keuzeletter* om het menu te selecteren.

Aanvullende aanpassingen

In Hoofdstuk 1 worden de belangrijkste nieuwe taalfuncties in dBASE beschreven. Bestudeer dit hoofdstuk om ideeën op te doen. Hierna ziet u een aantal voorbeelden van specifieke aanpassingen die u kunt aanbrengen:

- Gebruik de meegeleverde stuurelementen uit KNOPPEN.CC in de subdirectory VOORBD om knoppen met vooraf gedefinieerde pictogrammen voor algemene taken weer te geven (bijvoorbeeld **OK**, **Annuleren** of **Help**). Zie Hoofdstuk 15 voor meer informatie over het maken en gebruiken van eigen dBASE- en VBX-stuurelementen.
- Met de volgende nieuwe functies geeft u dialoogvensters weer waarin gebruikers gegevens kunnen invoeren:
  - CHOOSEPRINTER(). Hiermee geeft u een dialoogvenster weer om een printer te selecteren. De functie resulteert in de geselecteerde printer.
  - GETCOLOR(). Hiermee kunt u een dialoogvenster weergeven om een eigen kleur te definiëren of een kleur in het kleurpalet te selecteren.
  - GETDIRECTORY(). Hiermee geeft u een dialoogvenster weer om een directory te selecteren voor volgende commando's.
  - GETEXPR(). Hiermee geeft u het hulpmiddel Uitdrukking samenstellen weer. De functie resulteert in de gemaakte uitdrukking.
  - GETFILE() en PUTFILE(). Hiermee geeft u dialoogvenster weer om respectievelijk een bestaande bestandsnaam of een nieuwe bestandsnaam te selecteren of in te voeren.
  - GETFONT(). Hiermee geeft u een dialoogvenster weer om een tekenfont te selecteren. De functie resulteert in een reeks die bestaat uit de fontnaam, de tekengrootte, de fontstijl (als u een andere stijl selecteert dan Normaal) en de fontfamilie.
- Als u .PRS-bestanden met SQL-opdrachten gebruikt in uw applicatie, vervangt u de bijbehorende commando's USE door de functie SQLEXEC(). Zie Hoofdstuk 23 voor meer informatie over SQL-opdrachten.

Appendix



Werken met tekensets en taalaansturing

dBASE is een internationaal georiënteerd software-pakket dat een verscheidenheid aan talen ondersteunt. Elke taal of taalcategorie gebruikt een eigen tekenset en een eigen manier om de tekens in de set te sorteren en relateren. dBASE biedt uitgebreide ondersteuning voor talen en tekensets met meerdere taalaansturingsprogramma's en automatische conversie van OEM naar ANSI en omgekeerd.

Gebruikers die nog geen ervaring hebben met de Windows-omgeving, moeten weten wat het verschil is tussen de OEM- en ANSI-tekensets. Gebruikers die gegevens uitwisselen over land- of taalgrenzen heen, moeten weten hoe in dBASE taalaansturingsprogramma's worden gebruikt om gegevens in verschillende talen te verwerken.

In dit hoofdstuk wordt beschreven hoe de OEM- en ANSI-tekensets werken in dBASE en hoe u taalaansturingsprogramma's gebruikt.

Werken met tekensets

In het begin van de jaren tachtig stelden de ontwikkelaars van de IBM PC een geordende lijst van symbolen samen die de *uitgebreide IBM-tekenset* wordt genoemd. Deze lijst bevatte alle klassieke 7-bits ASCII -tekens en diverse rekenkundige symbolen, lijn- en kadertekens en tekens met accenten.

Hoewel deze tekenset voldeed voor bepaalde Engelstalige landen, schoot de set tekort voor de meeste andere landen. Diverse Europese talen maken bijvoorbeeld gebruik van tekens met accenten die niet voorkomen in de uitgebreide IBM-tekenset. Het gevolg was dat er ook andere tekensets werden ontwikkeld. Elke tekenset, ook de oorspronkelijke uitgebreide IBM-tekenset, is opgeslagen in een *codetabel*. Elke codetabel is ontwikkeld voor een bepaald land of een bepaalde groep landen en wordt aangegeven met een nummer van drie cijfers.

MS-DOS ondersteunt onder meer de volgende codetabellen:

- 437 De Engelse taal en bepaalde Westeuropese talen
- 850 De meeste Westeuropese talen
- 852 Veel Oosteuropese talen
- 860 Portugees
- 863 Frans-Canadees
- 865 Scandinavische talen

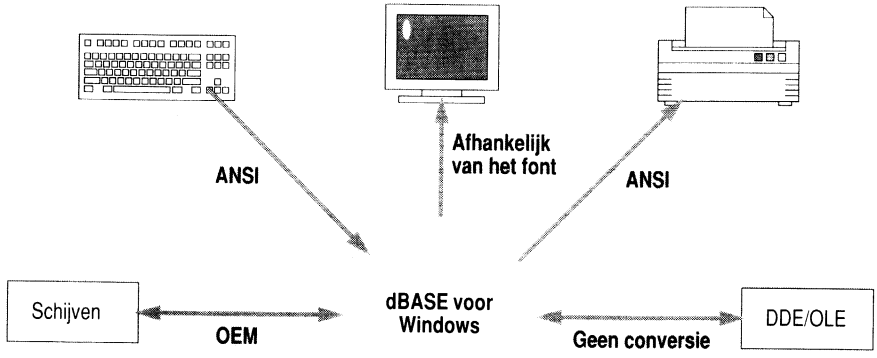
Deze codetabellen worden *OEM-codetabellen* (Original Equipment Manufacturer) genoemd. De klassieke uitgebreide IBM-tekenset is opgeslagen in OEM-codetabel 437 en is de standaardcodetabel voor de Verenigde Staten. Ook in Nederland zijn computers vaak ingesteld op codetabel 437, en anders vrijwel altijd op codetabel 850. Codetabel 850 wordt beschouwd als de standaardcodetabel voor de meeste Europese landen. Codetabel 850 bevat alle letters van codetabellen 437, 860, 863, en 865, maar niet alle symbolen. Veel kader- en lijntekens uit deze codetabellen zijn namelijk weggelaten in codetabel 850 om plaats te maken voor tekens met accenten.

Elk teken in een codetabel wordt aangegeven met een nummer. Dit decimale of hexadecimale nummer wordt een *codepunt* genoemd. De codepunt van het numerieke teken 4 is bijvoorbeeld 52 (decimaal) of 34 (hexadecimaal) in codetabellen 437 en 850.

De Windows-omgeving gebruikt een eigen tekenset, die de *ANSI-tekenset* wordt genoemd. Hoewel deze tekenset veel tekens gemeen heeft met de OEM-codetabellen, zijn de meeste lijntekens en rekenkundige symbolen uit deze codetabellen weggelaten. Daarnaast hebben veel tekens die ook voorkomen in OEM-codetabellen, vaak andere codepuntnummers in de ANSI-tekenset.

In dBASE worden de OEM-tekensets intern gebruikt, om de achterwaartse compatibiliteit zo groot mogelijk te maken. Wanneer tekens worden gelezen van schijf of worden geschreven naar schijf, wordt de tekenset niet geconverteerd. Omdat deze dBASE-versie een Windows-toepassing is, wordt de ANSI-tekenset gebruikt voor andere invoer- en uitvoerbewerkingen. In Afbeelding C.1 ziet u een overzicht van de OEM- en ANSI-conversie in dBASE:

Afbeelding C.1 Overzicht van OEM- en ANSI-tekensetconversie



Opmerking Wanneer er uitvoer wordt verzonden naar het scherm in dBASE, is de tekenset afhankelijk van het gebruikte font. In de volgende schermgebieden wordt bijvoorbeeld standaard de OEM-tekenset gebruikt:

- Tekst-editor (ook in memobestanden)
- Tekstvakken
- Commandovenster

In de volgende schermgebieden wordt echter standaard een ANSI-font gebruikt:

- Tabelrecordsvenster
- Labels voor knoppen
- Navigator

Werken met taalaansturingsprogramma's

In dBASE worden taalaansturingsprogramma's gebruikt om op te geven welke tekenset moet worden gebruikt en welke regels gelden voor de opgegeven tekenset. Elke tekenset komt overeen met een van de OEM-codetabellen. Het Frans-Canadese taalaansturingsprogramma maakt bijvoorbeeld gebruik van een tekenset die identiek is aan codetabel 863, terwijl het standaardaansturingsprogramma voor de Verenigde Staten gebruik maakt van een tekenset die identiek is aan codetabel 437. Het taalaansturingsprogramma dta het meest ingesteld zal worden in de Nederlandse versie van dBASE maakt gebruik van een tekenset die identiek is met codetabel 437. Het is belangrijk te weten dat dBASE gebruik maakt van deze interne codetabellen in plaats van de DOS-codetabellen.

De taalaansturingsprogramma's van dBASE bevatten tabellen waarmee de volgende onderdelen voor een bepaalde tekenset worden gedefinieerd of bestuurd:

- Alfabetische tekens
- Regels voor hoofdletters en kleine letters
- Sorteervolgorde voor sorteren of indexeren
- Reeksvergelijkingen (=, <, >, <=, >=)
- SOUNDEX-waarden (waarden die de fonetische spelling aangeven wanneer de exacte spelling niet bekend is)
- Regels voor de conversie tussen OEM- en ANSI-tekensets

In dBASE wordt elk aansturingsprogramma aangegeven met een tekenreeks die een *interne naam* wordt genoemd. De interne naam van het Nederlandse taalaansturingsprogramma voor codetabel 850 is bijvoorbeeld DB850NL0 en de interne naam van het Finse taalaansturingsprogramma is DB437FI0. In Tabel C.1 ziet u de Europese taalaansturingsprogramma's die beschikbaar zijn in dBASE.

Tabel C.1 Europese taalaansturingsprogramma's

Taal of land	Codetabel	Interne naam
Portugees/Brazilië	850	DB850PT0
Portugees/Portugal	860	DB860PT0
Deens	865	DB865DA0
Fins	437	DB437FI0
Frans	437	DB437FR0
Frans	850	DB850FR0
Frans/Canada	850	DB850CF0
Frans/Canada	863	DB863CF1
Duits	437	DB437DE0
Duits	850	DB850DE0
Italiaans	437	DB437IT0
Italiaans	850	DB850IT1
Nederlands	437	DB437NL0
Nederlands	850	DB850NL0
Noors	865	DB865NO0
Spaans	437	DB437ES1
Spaans	850	DB850ES0
Zweeds	437	DB437SV0
Zweeds	850	DB850SV1
Engels/G.B.	437	DB437UK0
Engels/G.B.	850	DB850UK0
Engels/V.S.	437	DB437US0
Engels/V.S.	850	DB850US0

Wanneer gegevens worden geconverteerd van OEM naar ANSI en omgekeerd, worden de meeste alfabetische tekens zonder problemen omgezet, omdat deze tekens voorkomen in zowel de OEM-codetabel als de ANSI-tekenset. De meeste uitgebreide

grafische symbolen in een OEM-codetabel komen echter niet voor in de ANSI-tekenset. In dit geval wordt het symbool omgezet in het teken dat er het dichtste bij komt, waardoor er gegevensverlies kan optreden.

Als u dus dBASE IV-bestanden gebruikt in een dBASE-sessie, moet u zorgen dat het huidige taalaansturingsprogramma overeenkomt met het taalaansturingsprogramma dat werd gebruikt in dBASE IV toen de bestanden werden gemaakt. In principe moet u zoveel mogelijk hetzelfde taalaansturingsprogramma gebruiken, zelfs wanneer u alleen tabellen gebruikt die zijn gemaakt in dBASE voor Windows-sessies.

Instellingen in de Nederlandse taalaansturing

Bij de installatie van dBASE wordt automatisch de taalaansturing gekozen voor de codetabel die is ingesteld in DOS en voor het land dat is ingesteld in de sectie **Internationaal** van het Configuratiescherm van Windows. Voor de Nederlandse versie van dBASE is dit meestal taalaansturingsprogramma DB437NL0. Hieronder worden de instellingen van dit taalaansturingsprogramma besproken.

Alfabetische tekens

Naast alle letters zonder accenten worden ook de volgende tekens als alfabetische tekens beschouwd:

- Hoofdletters: Ä Å Ç É Ñ Ö Ü
- Kleine letters: á à â ä å ç é è ê ë í ì î ï ñ ó ò ô ö ú ù û ü y ÿ

Hoofdletters en kleine letters

Bij het omzetten van kleine letters naar hoofdletters (commando UPPER) en omgekeerd (commando LOWER), wordt omgezet naar dezelfde hoofdletter of kleine letter, tenzij deze niet beschikbaar is. å wordt dus Å en ü wordt ü. Alleen bij een aantal kleine letters is geen hoofdletter beschikbaar:

á, à, â worden omgezet naar A, è, ê, ë naar E, í, ì, î, ï naar I, ó, ò, ô naar O, ú, ù, û naar U, ÿ naar Y.

Bij het omzetten van deze kleine letters gaat de oorspronkelijke waarde verloren. Als u een reeks met een van deze kleine letters eerst omzet naar hoofdletters en vervolgens weer naar kleine letters, resulteert dat in kleine letters zonder accenten.

```
hoofdl= upper("crèche")     && Resulteert in "CRECHE"
kleinel = lower(hoofdl)     && Resulteert in "creche"
```

Sorteervolgorde

Bij indexeren en sorteren wordt de gehanteerde volgorde bepaald door de taalaansturing. Afhankelijk van de opgegeven opties wordt al dan niet eerst gesorteerd op alle hoofdletters en dan op alle kleine letters. Als eerst wordt gesorteerd op hoofdletters, geldt de volgende volgorde:

< A Ä Å > B < C Ç > D < E É > F G H I J K L M < N Ñ > < O Ö > P Q R
S T < U Ü > V W X Y Z

< a á à â ä å > b < c ç > d < e é è ê ë > f g h < i í ì î ï > j k l m l m < n ñ >
< o ó ò ô ö > p q r s t < u ú ù û ü > v w x < y ÿ > z

Hierbij geven de punthaken aan dat een groep tekens, bijvoorbeeld c en ç, genest worden gesorteerd. Dat wilt zeggen dat eerst alle tekens in deze groep wordt verwerkt, en daarna pas het volgende teken wordt geëvalueerd. "Bruïne, de" komt dus na "Bruine, de", maar voor "Bruize, de" en voor "Brukken, van".

Als niet eerst wordt gesorteerd op hoofdletters, geldt de volgende volgorde:

< { a A } á à â { ä Ä } { å Å } > { b B } < { c C } { ç Ç } > { d D } < { e E } { é É } è ê ë >
{ f F } { g G } { h H } < { i I } í ì î ï > { j J } { k K } { l L } { m M } < { n N } { ñ Ñ } >
< { o O } ó ò ô { ö Ö } > { p P } { q Q } { r R } { s S } { t T } < { u U } ú ù û { ü Ü } >
{ v V } { w W } { x X } < { y Y } ÿ > { z Z }

Bij deze sorteermethode wordt genest op twee niveau's. Eerst worden de tekens in de groepen tussen accolades verwerkt, daarna worden in de groepen tussen punthaken de afzonderlijke tekens en de groepen tussen accolades verwerkt. "Een" komt dus na "een" en voor "één", maar "één" komt weer voor "eer".

Zoekacties en vergelijkingen

Bij zoekacties en andere vergelijkingsbewerkingen (bijvoorbeeld met de gelijkoperator =) bepaalt de instelling voor het commando SET EXACT in combinatie met de huidige taalaansturing hoe tekens met accenten worden verwerkt. Als SET EXACT is ingesteld op ON, moet een zoek- of vergelijkingsreeks altijd precies worden opgegeven. Als SET EXACT is ingesteld op OFF, wordt in de Nederlandse taalaansturing een teken met accent als identiek beschouwd met het bijbehorende teken zonder accent. In het volgende voorbeeld worden de reeksen "een" en "één" vergeleken met SET EXACT eerst ingesteld op ON en daarna op OFF.

```
set exact on                    && Preciese vergelijking opgeven
? "een" = "één"                 && Resulteert in .F. (onwaar)
set exact off                   && SET EXACT uitschakelen
? "een" = "één"                 && Resulteert in .T. (waar)
```

SOUNDEX-waarden

Met het commando SOUNDEX kunt u reeksen zoeken en vergelijken op basis van klankwaarden. De regels die worden gehanteerd bij dit commando worden bepaald door de taalaansturing. Zie SOUNDEX in de sectie "Taal" van Help voor een uitgebreide beschrijving van het commando en de regels ervoor in de Nederlandse taalaansturing.

Exacte en niet-exacte overeenkomsten bepalen

Wanneer u twee tekens vergelijkt in dBASE, kunt u naar een exacte overeenkomst (een *primaire overeenkomst*) of een niet-exacte overeenkomst (een *secundaire overeenkomst*) zoeken. U zoekt naar een exacte overeenkomst wanneer de voorwaarde EXACT is ingesteld op ON (aan) en een niet-exacte overeenkomst wanneer EXACT is ingesteld op

OFF (uit). Zie ook Hoofdstuk 20 voor interacties van SET NEAR en SET EXACT bij zoekbewerkingen.

Bij een exacte overeenkomst moeten de tekens precies hetzelfde zijn. De tekens A en Å lijken bijvoorbeeld op elkaar, maar voldoen niet aan het criterium voor een exacte overeenkomst. Deze tekens worden dan ook als verschillende tekens beschouwd in een SEEK- of FIND-uitdrukking. Bij een niet-exacte overeenkomst hoeven de tekens alleen maar in dezelfde algemene categorie te vallen. De tekens A en Å zijn gelijksoortig in veel talen, en voldoen aan het criterium voor een niet-exacte overeenkomst bij veel taalaansturingsprogramma's, waaronder het Nederlandse. Deze tekens worden dan ook als identieke tekens beschouwd in een SEEK- of FIND-uitdrukking. In het Amerikaanse taalaansturingsprogramma worden tekens met accenten bij zoekbewerkingen verwerkt op de codepuntwaarde in de codetabel, en moeten dergelijke tekens dus altijd precies worden opgegeven.

Exacte en niet-exacte overeenkomsten worden uitgevoerd met behulp van *primaire zwaarte* en *secundaire zwaarte*, die worden toegewezen aan elk teken door het taalaansturingsprogramma. Bij exacte overeenkomsten worden de primaire en secundaire zwaarte beide gebruikt, terwijl bij niet-exacte overeenkomsten alleen de primaire zwaarte wordt gebruikt en de secundaire zwaarte wordt genegeerd. U kunt bijvoorbeeld het commando SET EXACT gebruiken om aan te geven of u tekens met accenten wilt gelijkstellen aan de corresponderende tekens zonder accenten. Met de volgende commando's opent u het tabelbestand LANDEN.DBF en zoekt u een record met een sleutelwaarde die moet voldoen aan het criterium van een exacte overeenkomst:

```
SET EXACT ON
USE LANDEN ORDER TAG NAAM
SEEK "Italie"              && "Italië" wordt niet gevonden
```

EXACT is ON en de secundaire zwaarte van **e** en **ë** komt niet overeen, dus worden de tekens beschouwd als verschillende tekens. Met de volgende commando's opent u het tabelbestand LANDEN.DBF en zoekt u een record met een sleutelwaarde die moet voldoen aan het criterium van een niet-exacte overeenkomst:

```
SET EXACT OFF
USE LANDEN ORDER TAG NAAM
SEEK "Italie"              && "Italië" wordt wel gevonden
```

EXACT is OFF en de primaire zwaarte van **e** en **ë** komt overeen, dus worden de tekens beschouwd als identieke tekens.

Algemene taalaansturingsprogramma's gebruiken

Wanneer u een dBASE-sessie start, wordt automatisch een taalaansturingsprogramma geactiveerd. Dit aansturingsprogramma wordt het *algemene taalaansturingsprogramma* genoemd en wordt gebruikt bij alle volgende activiteiten in de sessie. Het algemene taalaansturingsprogramma bepaalt bijvoorbeeld hoe de FOR- en WHILE-uitdrukkingen worden geëvalueerd.

U kunt het algemene taalaansturingsprogramma op twee manieren instellen in dBASE:

- U geeft een taalaansturingsprogramma op bij de parameter **LANGDRIVER** op het tabblad **Aansturingen** in het dialoogvenster van het IDAPI-configuratieprogramma. Zie Hoofdstuk 1 in *Aan de slag* voor meer informatie over de instelling van een taalaansturingsprogramma voor IDAPI.
- U geeft een taalaansturingsprogramma op in de sectie [Settings] van het bestand DBASEWIN.INI (zie volgende sectie).

Als u geen taalaansturingsprogramma hebt opgegeven in DBASEWIN.INI, bepaalt de instelling in het IDAPI-configuratieprogramma het algemene taalaansturingsprogramma. Wanneer u een geldig aansturingsprogramma opgeeft in DBASEWIN.INI, vervangt dit aansturingsprogramma de instelling in het IDAPI-configuratieprogramma.

U geeft als volgt een algemeen taalaansturingsprogramma op in DBASEWIN.INI:

1 Sluit de huidige dBASE-sessie (als u een sessie hebt geopend).

2 Open het bestand DBASEWIN.INI in de subdirectory BIN van de installatiedirectory en plaats de volgende regel in de sectie [Settings]:

```
ldriver = <interne naam>
```

De interne naam van een Europees Spaans taalaansturingsprogramma voor codetabel 437 is bijvoorbeeld DB437ES1. U installeert dit aansturingsprogramma door de volgende regel in te voegen:

```
ldriver = DB437ES1
```

3 Sla uw wijzigingen op en start een nieuwe sessie in dBASE

U kunt het beste dezelfde onderliggende DOS-codetabel gebruiken als de interne codetabel in het actieve taalaansturingsprogramma. Het Spaanse taalaansturingsprogramma DB437ES1 gebruikt bijvoorbeeld een interne codetabel die overeenkomst met codetabel 437. De DOS-codetabel moet dan ook zijn ingesteld op 437. Er zijn diverse redenen om overeenkomende codetabellen te gebruiken. De belangrijkste reden heeft te maken met bestandsnamen: wanneer u een bestand opslaat in een dBASE-sessie met een bestandsnaam met tekens die niet bestaan in de huidige DOS-codetabel, wordt de bestandsnaam onleesbaar.

Raadpleeg uw DOS- en Windows-documentatie voor meer informatie over het wijzigen van DOS- en Windows-codetabellen.

Taalaansturingsprogramma's voor tabellen gebruiken

In dBASE wordt er automatisch een taalaansturingsprogramma toegewezen aan een tabel wanneer u de tabel maakt. Deze toewijziging wordt vastgelegd in het taalaansturings-ID (LDID) (een identificatie van 1 byte in de bestandsaanhef). Wanneer u een tabel maakt zonder hiervoor een andere tabel te gebruiken als basis, wordt het huidige algemene taalaansturingsprogramma toegewezen aan het taalaansturings-ID. Wanneer u een tabel maakt op basis van een andere tabel, wordt het algemene taalaansturingsprogramma of het taalaansturingsprogramma van de oorspronkelijke

tabel toegewezen aan het taalaansturings-ID van de nieuwe tabel. Dit is afhankelijk van het commando waarmee u het bestand maakt (zie Tabel C.2).

Tabel C.2 Commando's die taalaansturingsprogramma's (tabel of algemeen) voor nieuwe tabel

Algemeen taalaansturingsprogramma toewijzen aan LDID van de nieuwe tabel	Taalaansturingsprogramma van de oorspronkelijke tabel toewijzen aan de LDID van de nieuwe tabel
CREATE	COPY FILE
CREATE...FROM	COPY STRUCTURE
CREATE...STRUCTURE EXTENDED	COPY...STRUCTURE EXTENDED
	COPY TABLE
	COPY TO
	IMPORT
	JOIN
	MODIFY STRUCTURE
	RENAME TABLE
	SORT
	TOTAL

Met het volgende commando maakt u bijvoorbeeld een tabelbestand waarvoor het taalaansturings-ID wordt ingesteld op het huidige algemene taalaansturingsprogramma. In het voorbeeld is het algemene taalaansturingsprogramma ingesteld op DB850FR0 (Frans, codetabel 850) en is de tabel KLANTNL gemaakt met het Nederlandse taalaansturingsprogramma.

```
CREATE KLANTFR FROM KLANTNL     && Nieuwe tabel met Franse taalaansturing, overigens
                                && identiek met oude
```

Met de volgende commando's opent u een tabelbestand en maakt u een nieuw gesorteerd tabelbestand waarvoor het taalaansturings-ID wordt ingesteld op het taalaansturingsprogramma van de oorspronkelijke tabel:

```
USE CLIENTS                    && LDID bevat ander taalaansturingsprogramma dan
                                && huidige algemene taalaansturingsprogramma
SORT ON LASTNAME TO KLANTEN    && LDID van KLANTEN.DBF krijgt
                               && taalaansturingsprogramma van CLIENTS.DBF
```

Taalaansturingsprogramma en codetabel van tabel identificeren

Commando's reageren verschillend wanneer een taalaansturingsprogramma van een tabel afwijkt van het huidige taalaansturingsprogramma. U moet daarom kunnen bepalen welk taalaansturingsprogramma is toegewezen aan de LDID van een tabelbestand of welke codetabel wordt gebruikt door het taalaansturingsprogramma. Bestands- en veldnamen kunnen bijvoorbeeld geldig zijn in de ene taal maar niet in een andere, of een sleutelveld kan tekens bevatten die niet voorkomen in alle codetabellen van de taalaansturingsprogramma's.

Wanneer u een commando gebruikt waarmee een tabel wordt geopend en de tabel heeft een ander taalaansturingsprogramma dan het algemene taalaansturingsprogramma, wordt er een waarschuwingsdialoogvenster weergegeven. Deze fout start geen ON

ERROR-routine zoals in dBASE IV. Als u dit dialoogvenster niet wilt weergeven, schakelt u het venster uit door SET LDCHECK OFF uit te voeren.

Gebruik de functies LDRIVER( ) en CHARSET( ) om te bepalen welk taalaansturingsprogramma is opgegeven in de LDID van de tabel en welke codetabel wordt gebruikt door het taalaansturingsprogramma, bijvoorbeeld:

```
SET LDCHECK OFF                 && Automatische controle op comptabiliteit van
                                && taalaansturingsprogramma uitschakelen

USE AFNEMERS EXCLUSIVE
INDEX ON Bedrijf TAG Bedrijf
IF LDRIVER() = "DB437NL0"       && Als dit het Nederlandse taalaansturingsprogramma is...
   SEEK "Klück Systems"         && Zoeken
ELSE
   IF CHARSET() = "DOS:437"     && Als aansturingsprogramma codetabel 437 gebruikt...
      Waarsch1.Open()           && Waarschuwingsdialoogvenster openen
   ELSE                         && Als aansturingsprogramma niet codetabel 437 gebruikt...
      Waarsch2.Open()           && Ander waarschuwingsdialoogvenster openen
   ENDIF
ENDIF
```

Tabeltaalaansturingsprogramma's vs. algemene taalaansturingsprogramma's gebruiken

Wanneer het taalaansturingsprogramma van een tabel afwijkt van het huidige algemene taalaansturingsprogramma, wordt het tabelaansturingsprogramma automatisch geladen in het geheugen wanneer u de tabel opent. Sommige commando's gebruiken daarna het tabelaansturingsprogramma, terwijl andere het algemene taalaansturingsprogramma gebruiken.

Alle commando's die geen betrekking hebben op tabellen, gebruiken het algemene taalaansturingsprogramma. In Tabel C.3 ziet u de algemene regels voor bewerkingen op tabelgegevens waarbij het taalaansturingsprogramma afwijkt van het algemene taalaansturingsprogramma.

Tabel C.3 Context voor tabeltaalaansturingsprogramma's en algemene taalaansturingsprogramma's

Tabelaansturingsprogramma	Algemeen aansturingsprogramma
Uitdrukkingen in commando INDEX ON	Uitdrukkingen in commando SET FILTER
Bereikuitdrukkingen in FOR bij commando INDEX ON	Uitdrukkingen in FOR en WHILE bij elk commando behalve INDEX ON
Bereikcontroleuitdrukkingen in SET KEY	Uitdrukkingen in SET RELATION TO
Uitdrukkingen in commando SORT	
Secundaire overeenkomsten voor uitdrukkingen in LOOKUP( ), FIND, SEEK en SEEK( ), met EXACT is OFF	
Secundaire overeenkomsten voor uitdrukkingen in SET RELATION TO, met EXACT is OFF (het aansturingsprogramma van de subtabel wordt gebruikt)	

Als u bijvoorbeeld een tabelbestand maakt met het Nederlandse taalaansturingsprogramma, wordt een LDID geschreven naar de bestandsaanhef. Wanneer u dit bestand opent in een dBASE-sessie waarvoor het algemene taalaansturingsprogramma Engels is, bestaat er een conflict tussen de tekensets. Als u een index maakt met INDEX ON, wordt de logische volgorde van de index bepaald op basis van het tabeltaalaansturingsprogramma, bijvoorbeeld:

```
USE VOLK                        && Maken met Nederlandse taalaansturingsprogramma
INDEX ON NAAM TAG NAMEN         && Records sorteren op Nederlandse volgorde
```

Als u een filter maakt met SET FILTER, wordt de filtervoorwaarde echter bepaald op basis van het algemene taalaansturingsprogramma:

```
USE VOLK
SET FILTER TO NAMEN = "Indiers"  && Records uitsluiten met "Indiërs" in NAMEN
```

Incompatibele tekens in veldnamen verwerken

Wanneer u een tabelbestand gebruikt dat is gemaakt met een ander taalaansturingsprogramma dan het huidige algemene taalaansturingsprogramma, worden sommige tekens in het tabelbestand misschien niet herkend. Dit kan problemen veroorzaken.

Het Nederlandse taalaansturingsprogramma DB437NL0 heeft bijvoorbeeld dezelfde codetabel en tekenset als het Amerikaanse taalaansturingsprogramma DB437US0. Het Nederlandse taalaansturingsprogramma herkent echter uitgebreide tekens zoals **ü** en **É** als alfanumerieke tekens, die onbekend zijn in het Amerikaanse taalaansturingsprogramma. Wanneer een veldnaam dus dergelijke tekens bevat (zoals PRIVÉNAMEN in het volgende voorbeeld), ontstaan er problemen wanneer u probeert te verwijzen naar dit veld in een dBASE-sessie waarin het Amerikaanse taalaansturingsprogramma wordt gebruikt.

In dit geval wordt er een fout gegenereerd wanneer u het volgende commando uitvoert:

```
REPLACE PRIVÉNAMEN WITH "Eigen namen"
```

U kunt dit probleem oplossen door de veldnaam tussen scheidingstekens (: :) te zetten, zodat de veldnaam wordt beschouwd als een identificatie en de regels in het huidige taalaansturingsprogramma niet gelden:

```
REPLACE :PRIVÉNAMEN: WITH "Eigen namen"
```

Wanneer u dit commando gebruikt, wordt elk element in de veldnaam PRIVÉNAMEN behandeld als een identificatie (ook É) en wordt het commando goed uitgevoerd.

Index

Symbolen

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Borland

Borland, dat haar hoofdkantoor heeft in Amerika, heeft verder kantoren in Australië, België, Canada, Denemarken, Duitsland, Engeland, Frankrijk, Hong Kong, Italië, Japan, Korea, Maleisië, Nederland, Nieuw-Zeeland, Singapore, Spanje, Taiwan en Zweden. Nederland: Borland Benelux B.V., Postbus 71876, 1008 EB Amsterdam. België: Borland Belgium N.V., Boechoutlaan 55, Bus 1, 1853 Strombeek-Bever. • Part # DBS1150NL2177F • BOR 7119